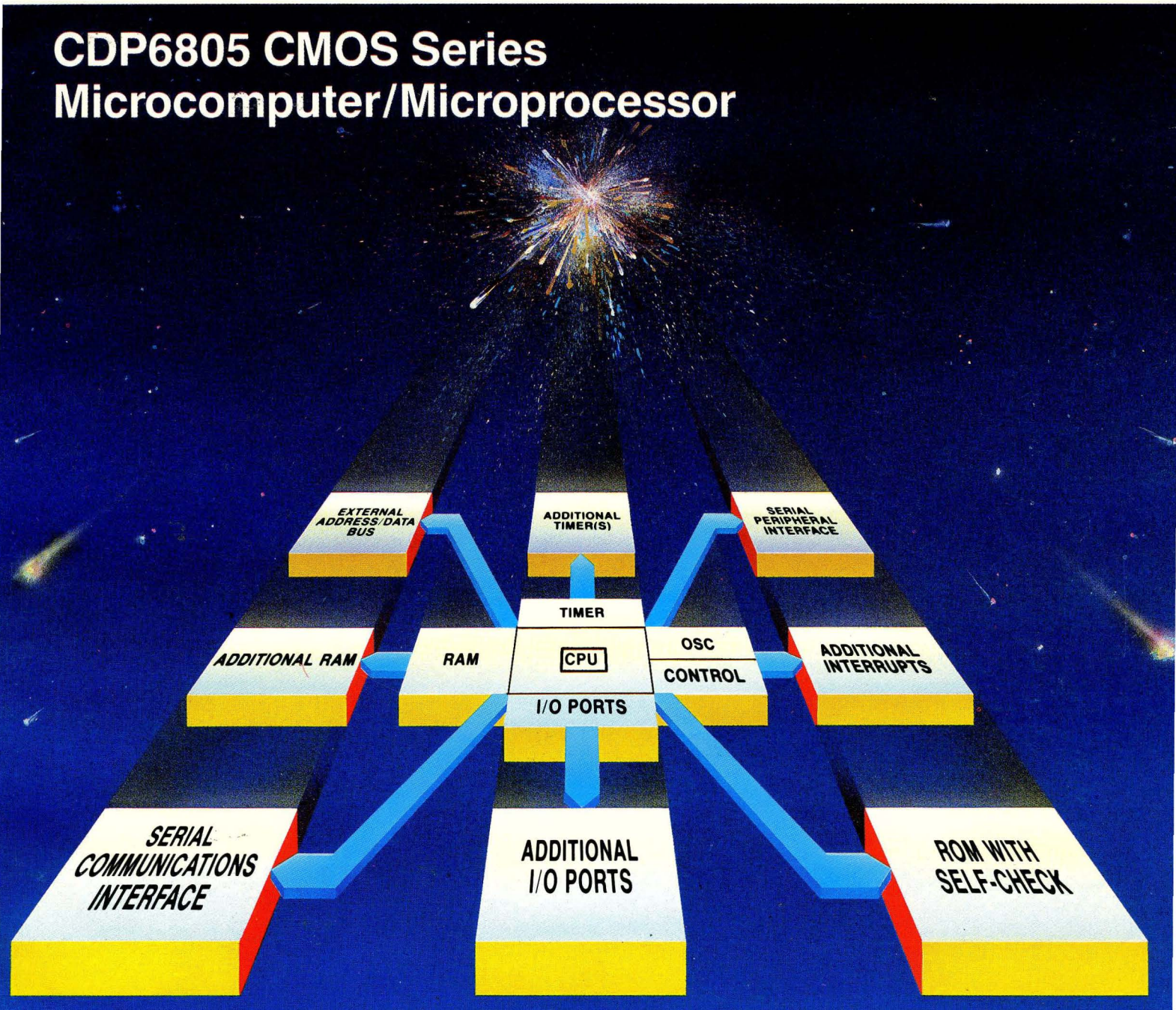
RCA

User Manual

CDP6805 CMOS Series Microcomputer/Microprocessor



User Manual for the RCA CMOS CDP6805-Series Microcomputers and Microprocessors

RCA Solid State | Somerville, NJ • Brussels • Paris • London
Hamburg • Sao Paulo • Hong Kong

Printed in USA 7/85

Foreword

The **RCA CDP6805-Series Microcomputers (MCUs) and Microprocessors (MPUs)** consist of the CDP6805E2 and CDP6805E3 MPUs and the CDP6805F2, CDP6805G2, CDP68HC05C4 and CDP68HC05D2 MCUs. This User Manual provides the system designer with a detailed guide to the CDP 6805-Series, describing the architecture and providing a set of simple, easy-to-use programming instructions. Examples are given to illustrate the operation and usage of each instruction. The CDP6805E2/E3 MPUs contain an 8-bit Arithmetic Logic Unit (ALU), accumulator, program counter, index register, stack pointer, condition code register, instruction decoder and timing and control logic. These MPUs are identical except that the directly accessible address space has been increased from 8K bytes on the E2 to 64K bytes on the E3.

The microcomputer versions CDP6805F2/G2 and CDP68HC05C4/D2 contain an on-chip oscillator, CPU, RAM, ROM, I/O and timer. The on-chip RAM permits these devices to operate without external memory. The addressing modes and register-like memory operations use this RAM to the fullest extent possible.

The CDP6805-Series features parallel I/O capability with each pin programmable as an input or output. The external interrupt input, and the capability for multiple nesting of subroutines and interrupts, are features usually found only on much more powerful architectures. These features permit the CDP6805-Series MCUs to be used in applications usually considered too complex for microcomputers. The external interrupt and counter/timer interrupt are vectored to different service routine addresses, which greatly simplifies interrupt programming. It also speeds execution of interrupt routines by eliminating software interrupt polling.

The CDP6805-Series devices have an on-chip counter/timer which greatly simplifies software development. The CDP6805E2/E3/F2/G2 have 8-bit counter/timers while the CDP68HC05C4/D2 have 16-bit counter/timers. The counter/timer can be used for timekeeping, measuring and generating pulses, and counting external events. The timer can also be set to "wake up" the MPU from the power-saving WAIT mode.

The lowest address spaces are reserved for memory-mapped I/O registers. The programmer may take full advantage of the versatile addressing modes and the register-like RAM operations of the family. User ROM sizes range from zero to greater than 4K bytes for the MPU. A self-check ROM is available on the CDP6805F2/G2 and on the CDP68HC05C4/D2. The ROM area used in the self-check operation is not included in the published ROM sizes. The user can get the entire ROM space for his program. A small portion of 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 be used for program storage as well.

The CDP6805 family includes types with either 64, 96, 112 or 176 bytes of on-chip RAM located in page zero. 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.

The CDP68HC05C4/D2 each have a built-in Serial Peripheral Interface (SPI) which is used to allow expansion while conserving I/O lines. In addition, the CDP68HC05C4 has a full UART-type internal Serial Communications Interface (SCI).

Table of Contents

Introduction

General

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 evolution 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 CDP6805 CMOS Family of microcomputers (MCU) and microprocessors (MPU). The CDP6805 Family is 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 because, 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 contain 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 CDP6805 CMOS Family of MCUs and MPUs are designed around a common core which consists of CPU, timer, oscillator, control section (for interrupts and reset), varying amounts of bidirectional I/O lines, and possibly ROM. In addition to this common core, additional items can be added such as additional memory and additional I/O lines. As of the printing of this manual, this versatile common-core design has already provided six different CDP6805 CMOS Family devices. The six different family members allow the user to choose the device best suited for his/her 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 CDP6805 CMOS Family members is included in this chapter. Detailed information concerning device operation is included in the following chapters as well as appendices. Data sheets on the individual processors are another source of information. 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 CDP6805 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 CDP6805 CMOS Family in a relatively short time. In some respects the CDP6805 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 CDP6805 CMOS and M6800 Families.

Architecture

The CDP6805 CMOS Family architecture has been optimized for controller applications rather than general purpose data processing operations. Several features contribute to this optimization:

Instruction Set

The instruction set used with the CDP6805 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 CDP6805 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 CDP6805 CMOS Family, when compared to other machines in this class, can be very misleading. A single CDP6805 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 CDP6805 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. An efficient use of pages zero and one would be for storage of tables because these two pages are easily accessed by the indexed addressing mode.

Addressing Modes

One of the chief measures of the effectiveness of a computer architecture is its ability to access data. The CDP6805 CMOS Family has several 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 CDP6805 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 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 6800 Family I/O devices, most CDP6805 CMOS Family I/O pins are individually programmed as inputs or outputs under software control.

Specific Features

The unique properties of CMOS technology (Complementary MOS with both P- and N-channel devices) are increasingly attractive to the needs of advanced microcomputer technology. Some applications are simply not feasible with PMOS, NMOS, or HMOS microcomputers.

Features and operating characteristics of the RCA CDP6805 Family of microcomputers and microprocessors which make them ideal choices include:

- Maximum power consumption of CMOS parts ranges from 1/15 to 1/200 of that of equivalent HMOS parts.

- The low power consumption of CMOS is important in several casses of applications, including:
  1) Portable equipment — hand-held and other portable units operating from self-contained batteries.
  2) Battery back-up — CMOS is appropriate in AC-powered applications when some or all system functions must continue during power outages. A small, rechargeable battery keeps a CMOS MCU operable.
  3) Storage batteries — Automotive and telephone equipment operate from large batteries. Automobile battery drain must be low when the engine is not running. Telephones must operate independently of AC power.
  4) Heat dissipation — Packaging constraints sometimes preclude dissipating electronics-generated heat, or the heat is costly to dissipate. In addition, dissipation of heat directly affects device reliability.
  5) 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 continually.
- Operation over a wide range of supply voltages. CMOS is used where the supply voltage fluctuates, such as in battery-powered equipment; or if line power is available, a low-cost, loosely regulated supply may be used.
- Fully static CMOS circuitry, no minimum clock frequency. CMOS microcomputers may be operated at any clock frequency less than the guaranteed maximum. This feature may be used to conserve power, because power consumption increases with higher clock frequencies. The CDP6805 Family features STOP and WAIT instructions to place the CPU in low power consumption modes. Static operation may also be advantageous during product development.

Hardware

Every CDP6805 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.

Hardware Common To All Devices

Figure 1 details the hardware functional blocks common to all CDP6805 CMOS Family devices.

The central processor unit (CPU) contains the 8-bit arithmetic-logic unit (ALU), accumulator, pro-

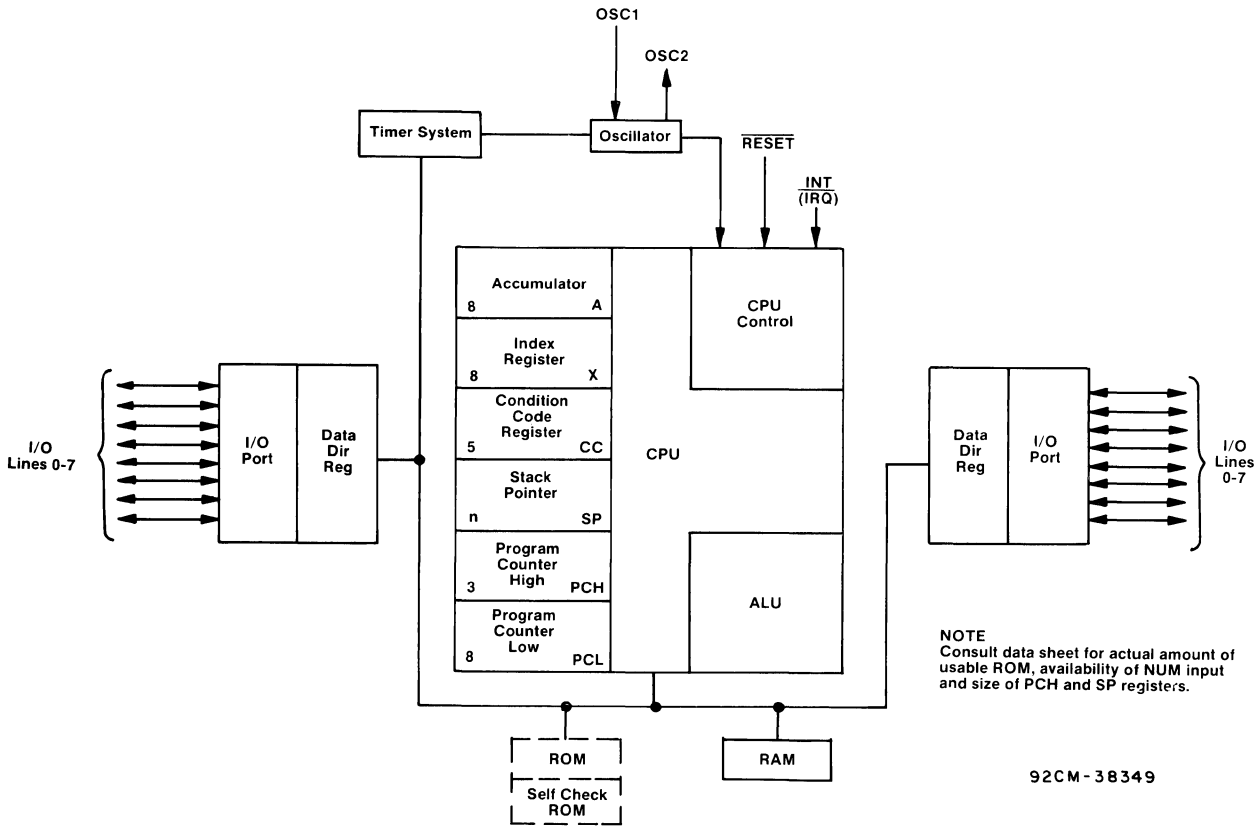


Figure 1 - CDP6805 CMOS Family Basic Microcomputer Block Diagram

gram 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 CDP6805 CMOS Family heritage.

The CDP6805 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.

All microcomputers in the family (CDP6805F2, CDP6805G2, CDP68HC05C4, and CDP68HC05D2) have on-chip user ROM and self-check programs.

Every member of the family has an on-chip oscillator. The oscillator on the microcomputers is mask-selectable as either crystal input or RC network.

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 subroutines and interrupts, are features usually found on much more powerful architectures. They permit a CDP6805 CMOS Family MCU to be used in projects usually considered too complex for microcomputers.

A feature which greatly simplifies software development and extends the capability of a microcomputer is an on-chip timer/counter. This counter and its prescaler can be programmed for innumerable functions. It can generate an interrupt at software-selected intervals. The timer/counter can also be used for timekeeping, measuring and generating pulses. The timer can be set to "wake-up" the processor from the power-saving WAIT mode, and those parts with an external timer input can be used to count external events.

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 processor addresses are reserved for memory-mapped I/O registers. The programmer of the CDP6805 CMOS Family may take full advantage of the versatile addressing modes and the register-like RAM operations of the Family.

Comparison of CDP6805 Family Members

This manual covers the six members of the CDP6805 CMOS Family: CDP6805E2, CDP6805E3, CDP6805F2, CDP6805G2, CDP68HC05C4, and CDP68HC05D2. The features of each are as follows:

- The CDP6805E2 and CDP6805E3 are 40-pin microprocessors. They feature, on-chip, 112 bytes of RAM, an oscillator, bidirectional I/O lines, and an 8-bit timer with software-programmable 7-bit prescaler. The CDP6805E2 and CDP6805E3 are identical except that the directly accessible address space has been increased from 8K on the E2 to 64K on the E3. To maintain the 40-pin package of the E2, the three additional required address lines were taken from the three most significant bits of Port A. Consequently the CDP6805E2 has 16 bidirectional I/O lines and the CDP6805E3 has 13 bidirectional I/O lines. Both parts have a multiplexed address and data bus.

- The CDP6805F2, CDP6805G2, CDP68HC05C4, and CDP68HC05D2 are microcomputers.

- The CDP6805F2 is a 28-pin microcomputer featuring, on-chip, 64 bytes of RAM, 1089 bytes of ROM, an oscillator, 16 bidirectional I/O lines, 4 unidirectional input lines, and an 8-bit timer with software-programmable 7-bit prescaler.

- The CDP6805G2 is a 40-pin microcomputer featuring, on-chip, 112 bytes of RAM, 2106 bytes of ROM, an oscillator, 32 bidirectional I/O lines, and an 8-bit timer with software-programmable 7-bit prescaler.

- The CDP68HC05C4 and CDP68HC05D2 are 40-pin microcomputers. In addition to the on-chip RAM, ROM, oscillator, and bidirectional I/O lines found on the other members of the family, the CDP68HC05C4 and CDP68HC05D2 contain a serial peripheral interface and a 16-bit programmable timer. In addition, the CDP68HC05C4 features a serial communications interface (SCI), and the CDP68HC05D2 features software-programmable open drain PORT A outputs, PORT B interrupt, wire "OR" mode for the SPI, software-programmable external oscillator timer input, and an on-chip timer oscillator.

Refer to Table I for a list of the members of the CDP6805 CMOS Family and their respective features.

Table I — Comparison Chart for CDP6805 Family Members

FEATURES	CDP6805E2	CDP6805E3	CDP6805F2	CDP6805G2	CDP68HC05C4	CDP68HC05D2
Technology	CMOS	CMOS	CMOS	CMOS	CMOS	CMOS
Number of Pins	40	40	28	40	40	40
On-Chip RAM (bytes)	112	112	64	112	176	96
External Address Space	8K	64K	—	—	—	—
On-Chip User ROM (bytes)	0	0	1089	2106	4160	2176
Bidirectional I/O Lines	16	13	16	32	24	24
Unidirectional I/O Lines	0	0	4 inputs	0	1 input	1 input
Timer Size	8	8	8	8	16	16
Serial Peripheral Interface	no	no	no	no	yes	yes
Serial Communications Interface	no	no	no	no	yes	no
Interrupts	External, Timer, SWI	External, Timer, SWI	External, Timer, SWI	External Timer, SWI	External, Timer, SCI, SPI, SWI	External, Timer, SPI, Port B SWI
Self Check Mode	no	no	yes	yes	yes	yes
On-Chip Oscillator	Crystal	Crystal	RC or Crystal	RC or Crystal	RC or Crystal	RC or Crystal
On-Chip Timer Oscillator	no	no	no	no	no	yes
Typical Full-Speed Operating Power at 5V	35mW	35mW	10mW	15mW	25mW	25mW
Typical WAIT Mode Power at 5V	5mW	5mW	3mW	4mW	7.5mW	7.5mW
Typical STOP Mode Power at 5V	25μW	25μW	25μW	25μW	5μW	5μW

Software Description

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/her system. Processors such as those of the CDP6805 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 CDP6805 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, because only a single address map must be supported, very few special-purpose instructions are necessary in the CDP6805 CMOS Family instruction set. The overall result 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.

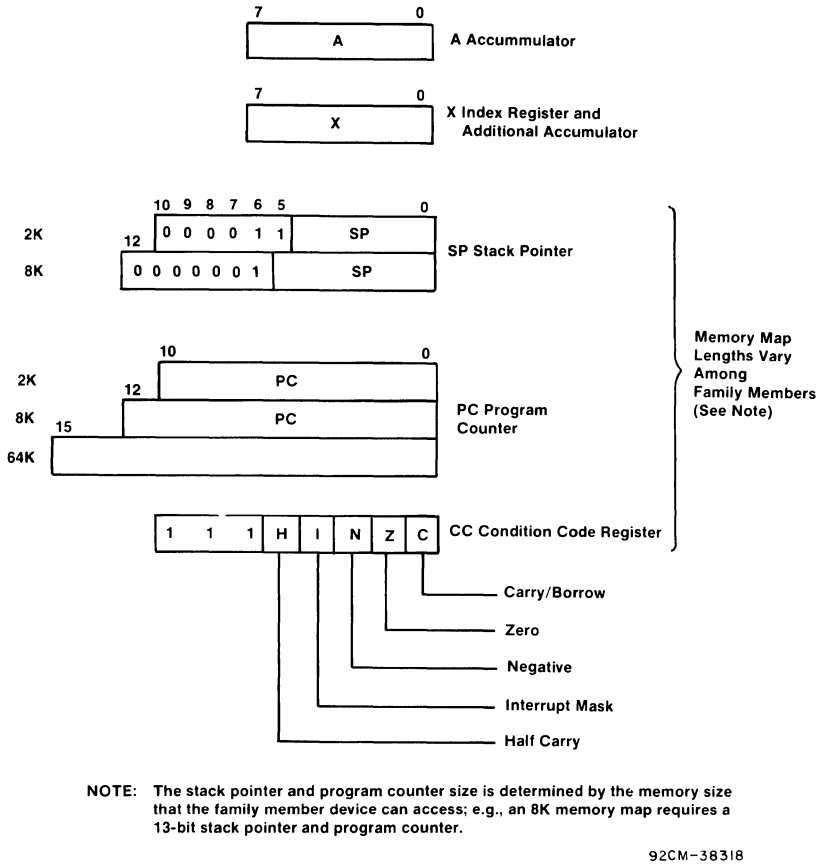
LDA — STA	Load and Store
INC — DEC	Increment and Decrement
BEQ — BNE	Branch if Equal and Branch if Not Equal
ADD — SUB	Add and Subtract
AND — ORA	Logic AND and Logic OR
BCLR — BSET	Bit Clear and Bit Set
ROR — ROL	Rotate Right and Rotate Left
JSR — RTS	Jump-To-Subroutine and Return-From-Subroutine

The symmetry provided by the CDP6805 CMOS Family instruction set means that the programmer need only remember 30 to 40 separate instructions to know the entire instruction set. The CDP6805 CMOS family has at least 61 instructions in its instruction set. The CDP68HC05C4 and CDP68HC05D2 microcomputers have an additional instruction to MULTIPLY.

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 the CDP6805 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 Opcode Map of Appendix A, the 61/62 separate instructions, enhanced by the seven addressing modes, expand into 209/210 opcodes; however, the programmer need only remember 68/69 items (61/62 instructions plus seven addressing modes) instead of 209/210.

Register Set

Each CDP6805 CMOS Family member contains five registers as shown in Figure 2. 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.



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.

92CM-38318

Figure 2 - CDP6805 CMOS Family Register Architecture

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 section 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

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 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 section.

The X register is also used in the CDP6805 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       Load Length into Index Register
REPEAT  LDA     SOURCE,X    Load Data from Memory at Source+Contents X into Accumulator
        STA     DESTIN,X    Store Data from Accumulator into Memory at Destination+Contents of X
        DECX                Set Up to Point to Another Cell, Also Control Count
        BNE     REPEAT      Repeat if More to Transfer
```

The X register is also useful in counting events because 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.

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

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 16 bits.

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. Because 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.

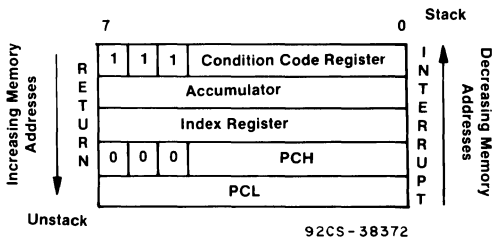
Because the CDP6805 is a family of devices, the actual size of the stack pointer may vary with memory size of the particular family member (see appropriate data sheets). But from the programmer's perspective, the stack pointers all appear similar on the different members. Both the hardware $\overline{\text{RESET}}$ pin and the reset stack pointer (RSP) instruction reset the stack pointer to its maximum value ($7F on the CDP6805E2/E3/F2/G2, $FF on the CDP68HC05C4/D2). The stack pointer on the CDP6805 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 CDP6805 CMOS Family).

Nested subroutine calls and interrupts must not underflow the SP. The usable stack length will vary between devices. In the CDP6805 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 because the previously stacked data will be lost. For a CDP6805 CMOS Family device, with a 6-bit stack pointer, the calculation is: 2^6 or 64 bytes maximum.

In the CDP6805 CMOS Family, the stack builds in the direction of decreasing address. The SP always points to the next empty location on the stack. The SP is decremented each time a data type 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, or during a reset pointer (RSP) instruction, the SP is set to its upper limit ($7F for the CDP6805E2/E3/F2/G2 and $FF for the CDP68HC05C4/D2).

The order in which bytes are stored onto and retrieved from the stack is shown in Figure 3. Notice

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 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. In the CDP6805 CMOS Family, PC fixed bits are always clear. The CDP6805E3 has no fixed bits in the program counter.

Figure 3 - Stacking Order

Condition Code Register (CC)

The CDP6805 CMOS Family uses five condition code flag bits, labeled H, I, N, Z, and C, which reside in the 8-bit 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 B. Any bit or combination of bits, except the I bit, is testable using the conditional branch instructions. See the Addressing Modes section for more information.

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 addition 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 a 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.

ZERO (Z). The Z bit is set if the result of the last arithmetic, logical, or data operation is zero. The 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 unless the eight resultant bits were all zeros (Z bit set).

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 is 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 based on the value of N. In using the CPX instruction, the N bit remains clear and no branch is taken, as long as the X register contents are 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 is set to one and a branch is initiated (using the BMI instruction).

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 because 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. The section on **Software Applications** describes a routine which uses the H bit to emulate the MC6800 DAA (decimal adjust) instruction.

INTERRUPT MASK (I). When the I bit is set, the external and timer interrupts 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 the program instructions. Some program instructions change the I bit explicitly, whereas others cause it to change implicitly. 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 CDP6805 CMOS Family parts also automatically set the I bit as part of instruction. See **Interrupts** in the **Hardware Features** section for more information.

NOTE: The SWI instruction and $\overline{RESET}$ are the only non-maskable interrupts in the CDP6805 CMOS Family.

Addressing Modes

The power of any computer lies in its ability to access memory. The addressing modes of the processor provide that capability. The CDP6805 CMOS Family has a powerful set of addressing modes.

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. Consequently, the addressing modes expand the basic 61 CDP6805 CMOS Family instructions into 209 separate operations. 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 CDP6805 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 $ 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 an immediate operand. Therefore the number is found in the location immediately following the opcode.

There are seven different addressing modes used in the CDP6805 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 subdivisions 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 E. This allows the processor bus activity and instruction operation relationship to be studied.

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 C as part of control and read/modify/write instruction tables.

The following is an example of a subroutine that clears the accumulator and index registers plus the C-bit and then returns. Figure 4 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
```

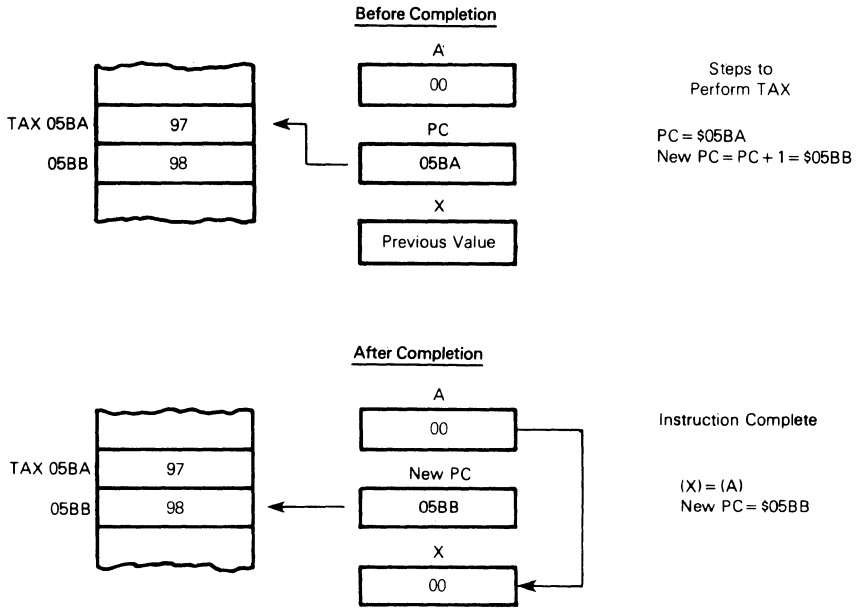


Figure 4 - Inherent Addressing Mode Example

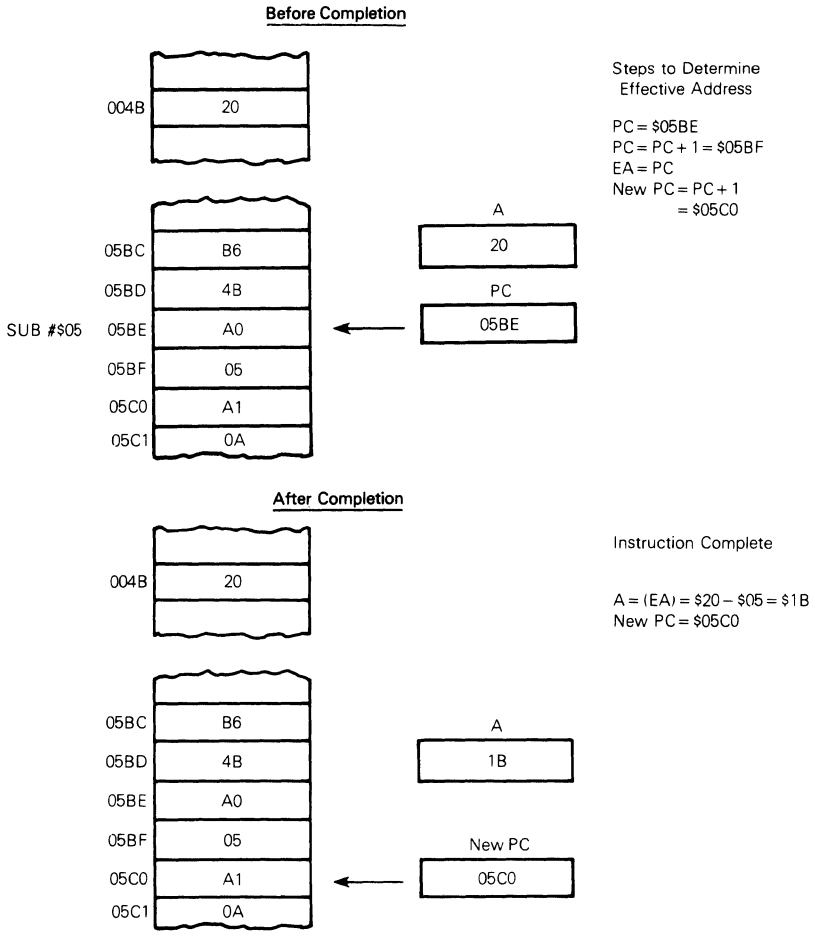


Figure 5 - Immediate Addressing Mode Example

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 C.

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 the number 10. Figure 5 shows an example of the steps required to perform the SUB instruction.

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

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 CDP6805 CMOS Family memory space, I/O, RAM, and ROM. The extended addressing mode allows an instruction to access all of memory. Extended addressing mode instructions are three bytes long, a one-byte opcode plus a two-byte address. All register/memory instructions, as shown in Appendix C, 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 6 shows an example of the steps required to determine the EA of the location containing the data to be loaded into the index register.

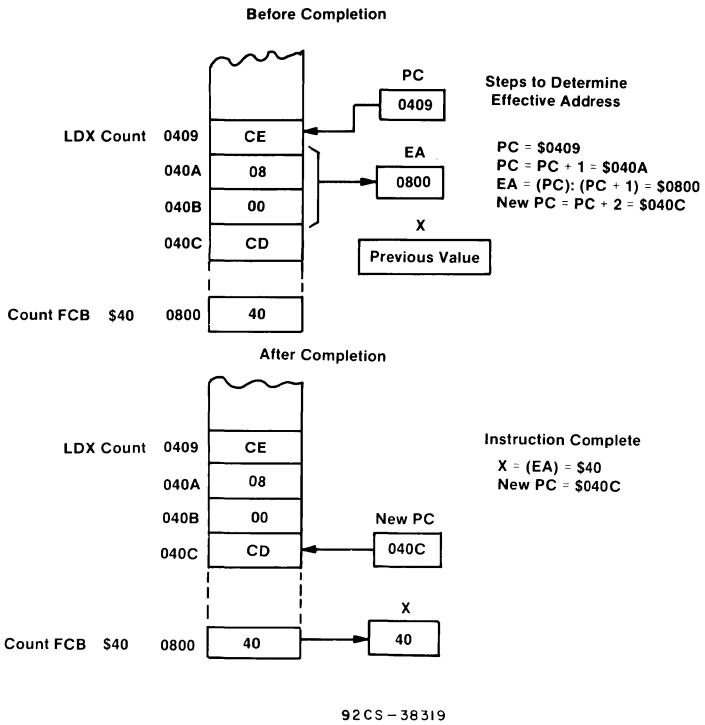


Figure 6 - Extended Addressing Mode Example

```
                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
```

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 access any location in page zero (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 7 illustrates the steps required to determine the EA of the most significant byte of the first number, the contents of which is loaded into the accumulator.

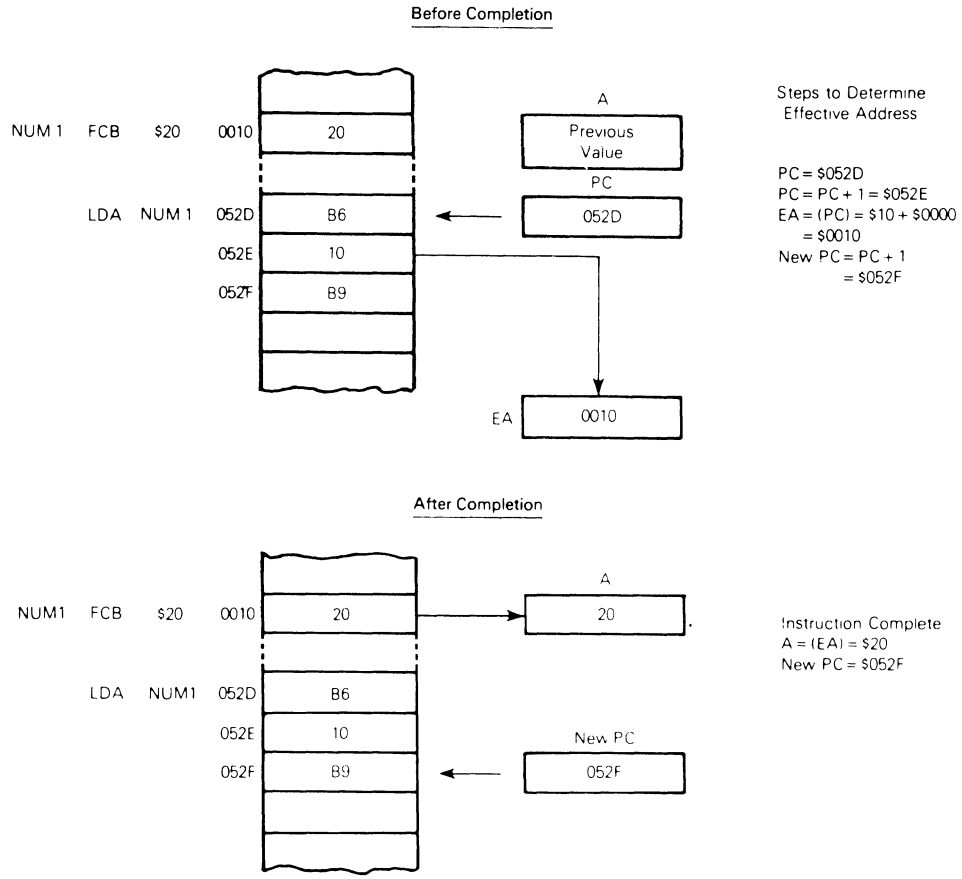


Figure 7 - Direct Addressing Mode Example

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

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 CDP6805 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.

Indexed — No Offset. In this mode the contents of the X register are 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 8 illustrates the steps necessary to determine the EA of a memory location pointed to by the index register. The contents of the accumulator are stored into this memory location.

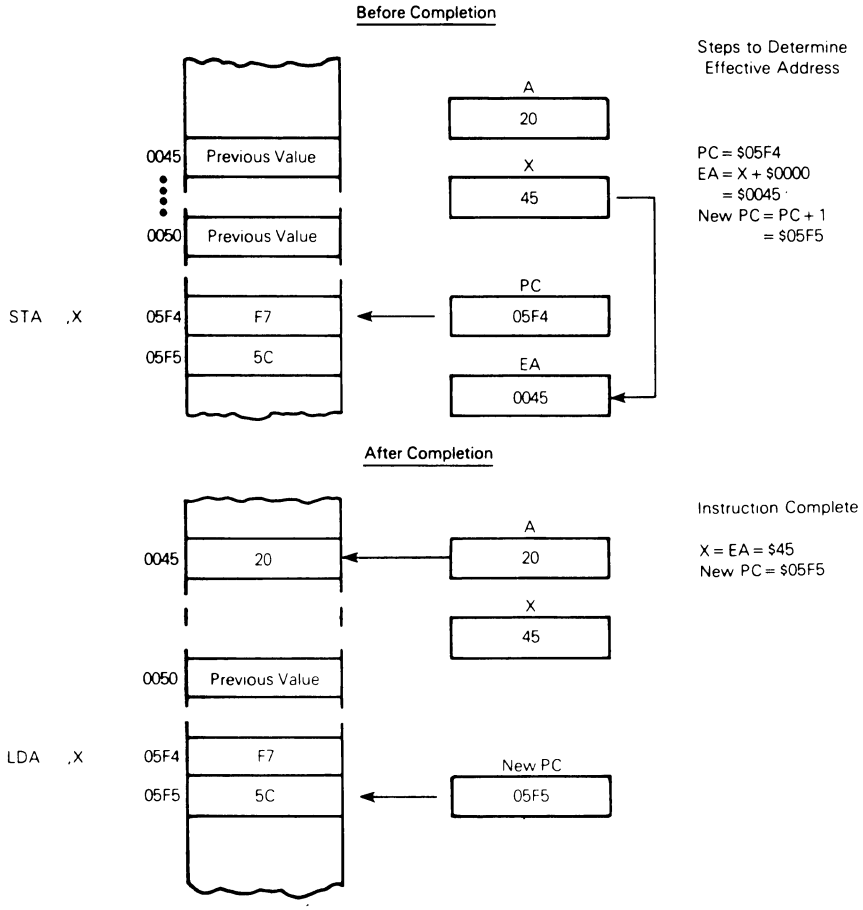


Figure 8 - Indexed Addressing Mode, No Offset Example

```
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
```

Indexed — 8-Bit Offset. To determine the EA in this addressing mode, the contents of the X register are 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 CDP6805 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 match one of the list items. Figure 9 shows the steps required to determine the EA of the next item to be compared.

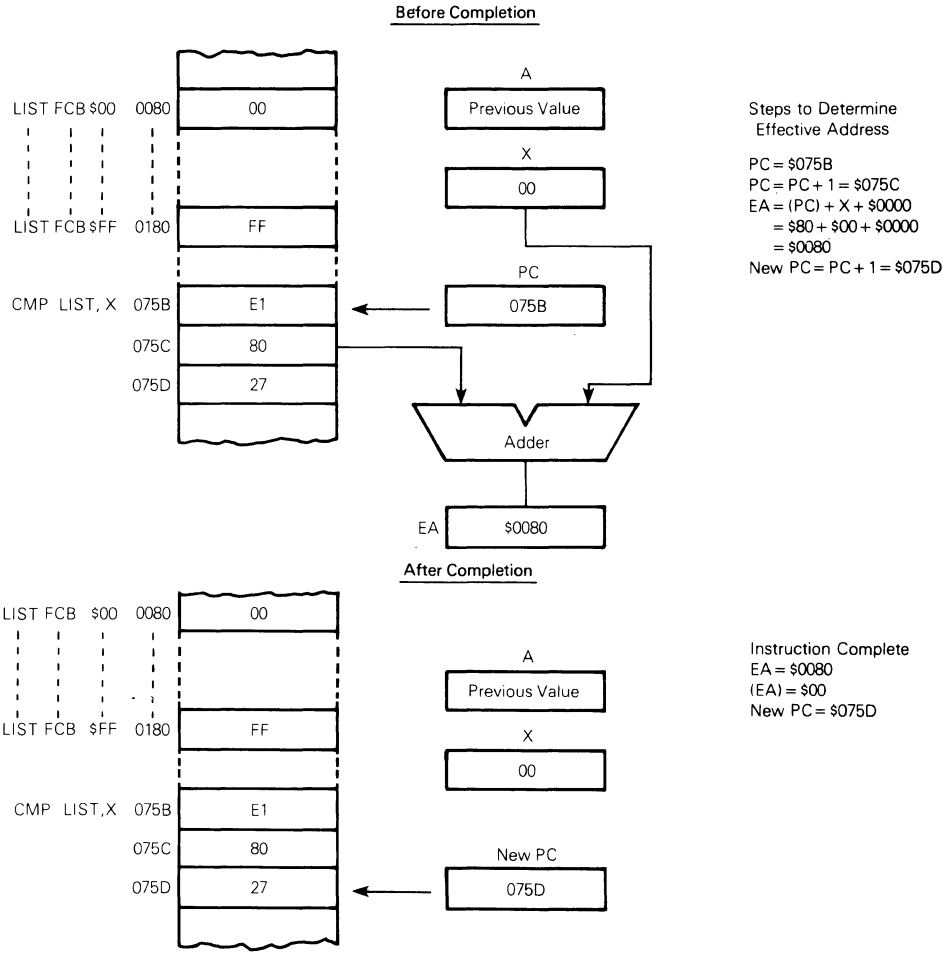


Figure 9 - Indexed Addressing Mode, 8-bit Offset Example

```
                    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 Done Else Repeat
0672  81            RETURN  RTS
```

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 opcode 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 because the offset is 16 bits, the tables being referenced can be anywhere in the CDP6805 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

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 10 illustrates the steps required to determine the EA from which to store the memory address contents into the accumulator.

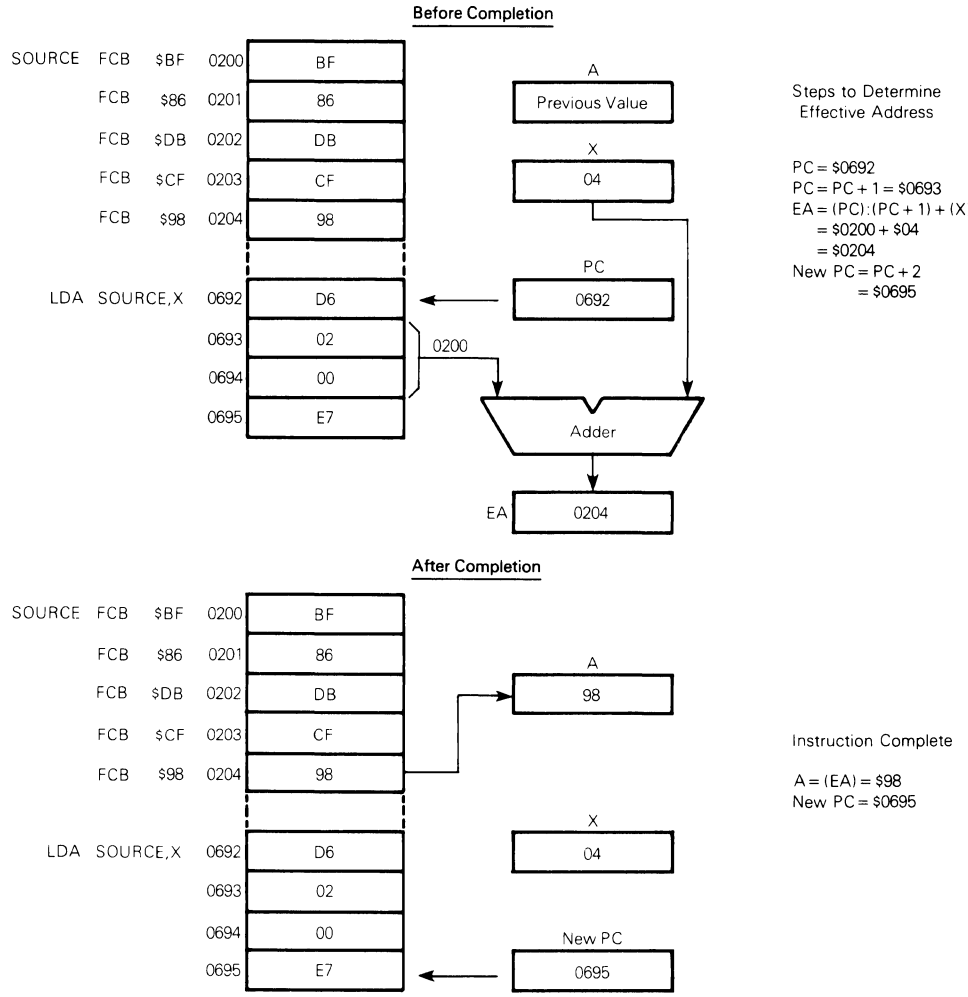


Figure 10 - Indexed Addressing Mode, 16-Bit Offset Example.

```
                    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 to Transfer
```

Indexing Compatibility. Because the index register in the CDP6805 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 CDP6805 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.

Relative Addressing Modes

Relative addressing is used only for branch instructions and specifies a location relative to the current value of the PC. The EA is formed by adding the contents of the byte following the opcode to the value of the PC. Because 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. Notice 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 <= R <= (PC+2) + 127
or
PC-126 <= R <= PC + 129 (for conditional branch only)

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

PC + 1 → PC
(PC) → TEMP
PC + 1 → PC
EA = PC + TEMP iff branch is taken

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 11 contains an instruction to branch to REPEAT, if the condition code register Z bit has not been set by the previous program step (DECX). Notice in Figure 11 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. Notice in this case the relative offset byte $FA indicates a backward branch because 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)
```

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.

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.

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

The following example compares the true bit manipulation of the CDP6805 CMOS Family to the conventional method of bit manipulation. This example uses the bit manipulation instruction to turn off an 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.

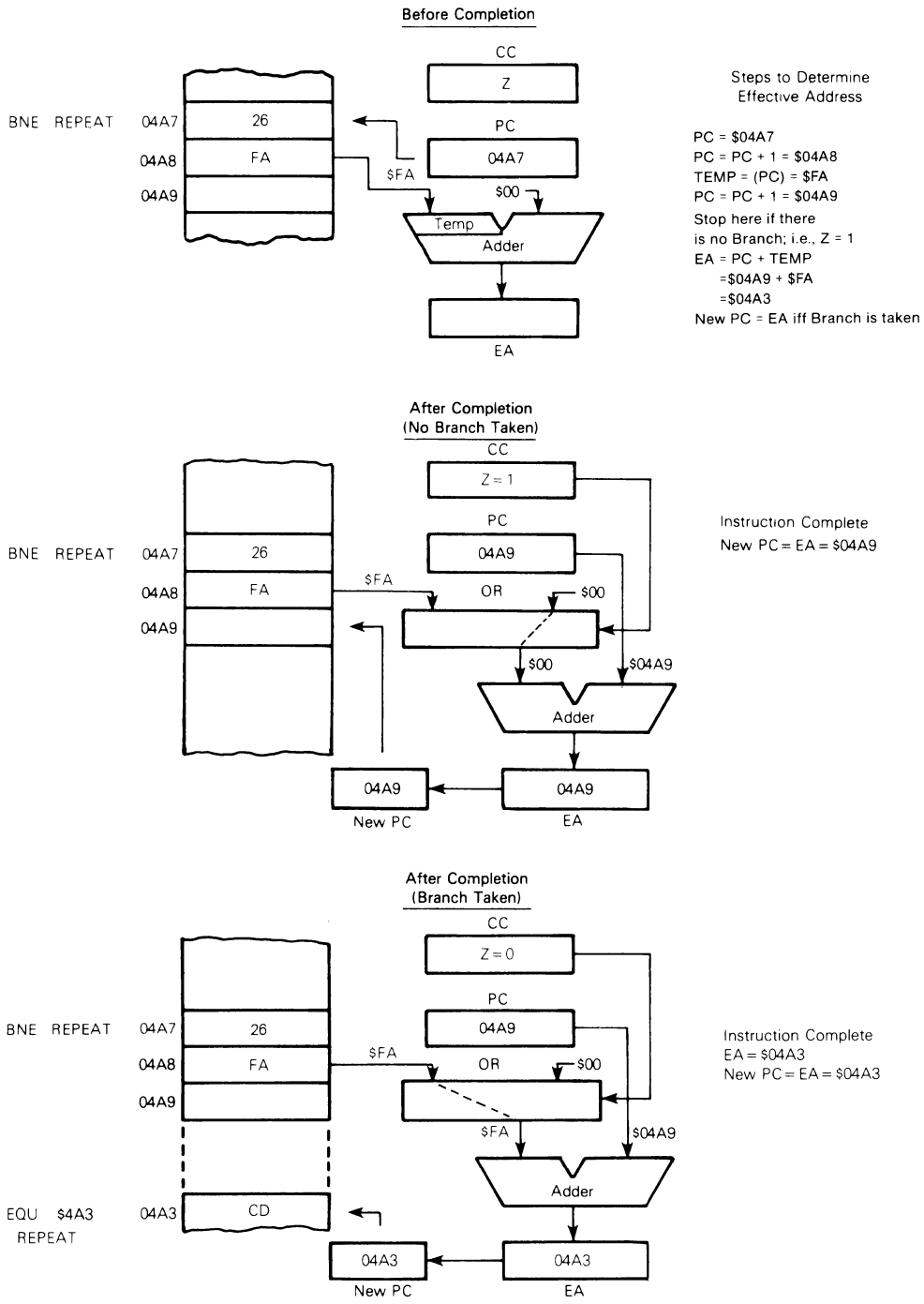


Figure 11 - Relative Addressing Mode Example

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
90594   14  01              BSET    2,PORTB     Turn On LED if Timer Times Out

                        CONVENTIONAL INSTRUCTIONS

                        LDA     PORTB       Get Port B Data
                        AND     #$FB        Mask Out Proper Bit
                        STA     PORTB       Save Updated Data
                REPT    LDA     TIMER       Loop Until TCR is Set
                        BIT     #$80
                        BNE     REPT
                        LDA     PORTB       Turn On LED
                        ORA     #$04
                        STA     PORTB
```

Figure 12 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.)

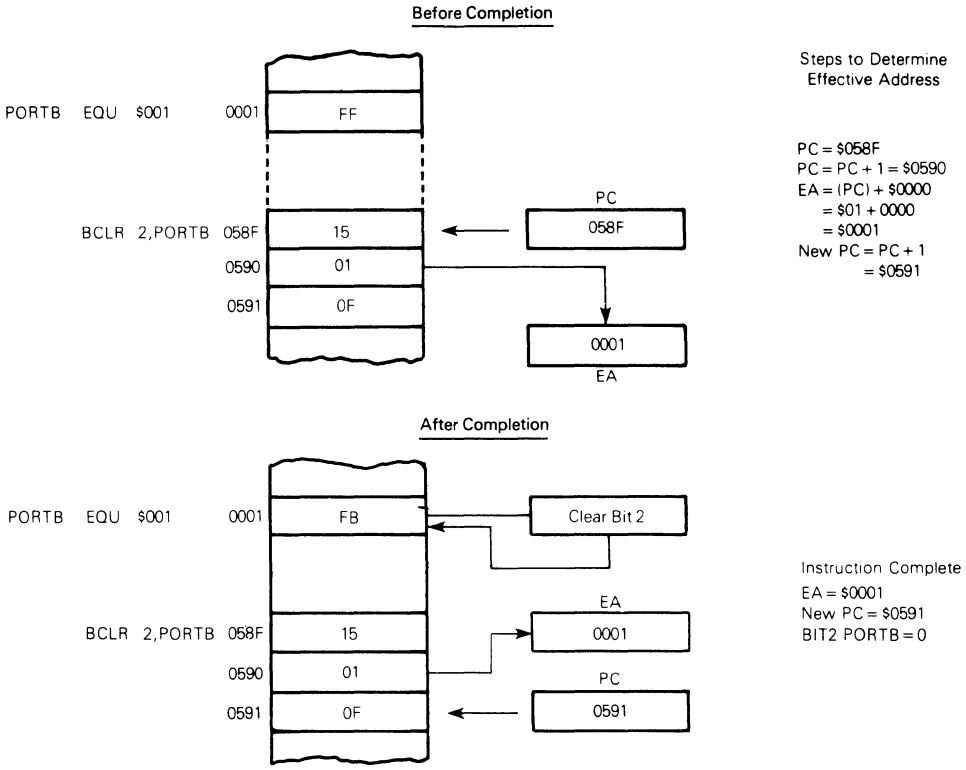


Figure 12 - Bit Set/Clear Addressing Example

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 <= R <= 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 BRCLR 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 an LED. Figure 13 illustrates this loop by showing both the branch and no branch status. Notice 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.

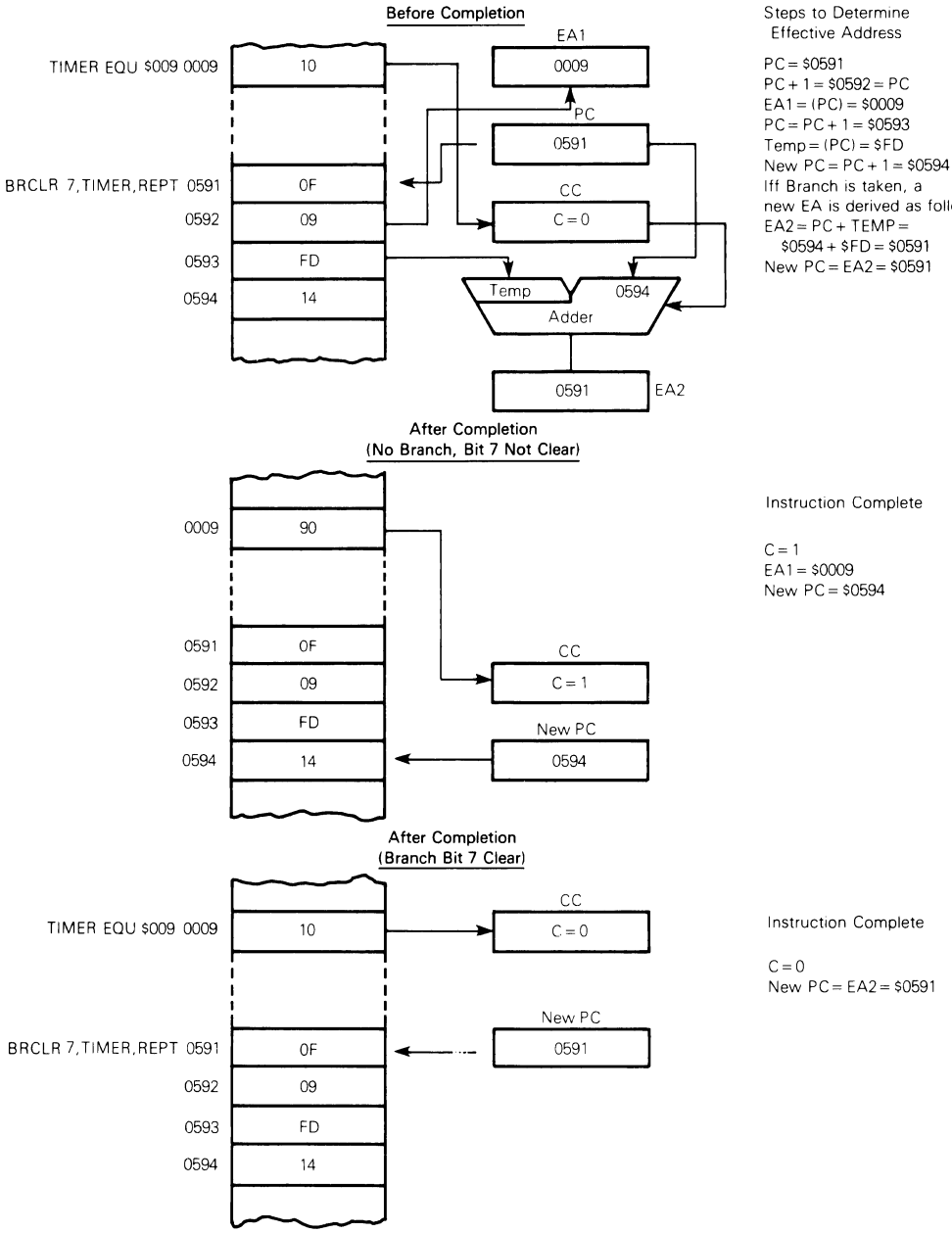


Figure 13 - Bit Test and Branch Addressing Mode Example

Instruction Set Overview

Introduction

It is convenient to view the CDP6805 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. A detailed definition of the instruction set used with the CDP6805 CMOS Family is included in this document. Appendix B contains an alphabetical listing of the instruction set; Appendix C provides a tabular functional listing of the instruction set; Appendix D contains a numerical listing which shows the mnemonic, addressing mode, cycles, and byte of the instruction set; Appendix E provides a cycle-by-cycle summary of the instruction set; and Appendix F contains an instruction set opcode map.

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 provided with store and jump instructions (STA, STX, JMP, and JSR). An alphabetical listing of the register/memory instructions 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 from Accumulator

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. Notice 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 (Two's Complement)
ROL	Rotate Left thru Carry
ROR	Rotate Right thru Carry
TST	Test for Negative or Zero

The multiply instruction MUL, available on the CDP68HC05C4 and CDP68HC05D2 microcomputers, is also in the read/modify/write class.

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

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)

Branch Instructions

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.

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

Notice that the BIH and BIL instructions permit an external interrupt pin ($\overline{INT}$ or $\overline{IRQ}$) to be easily tested.

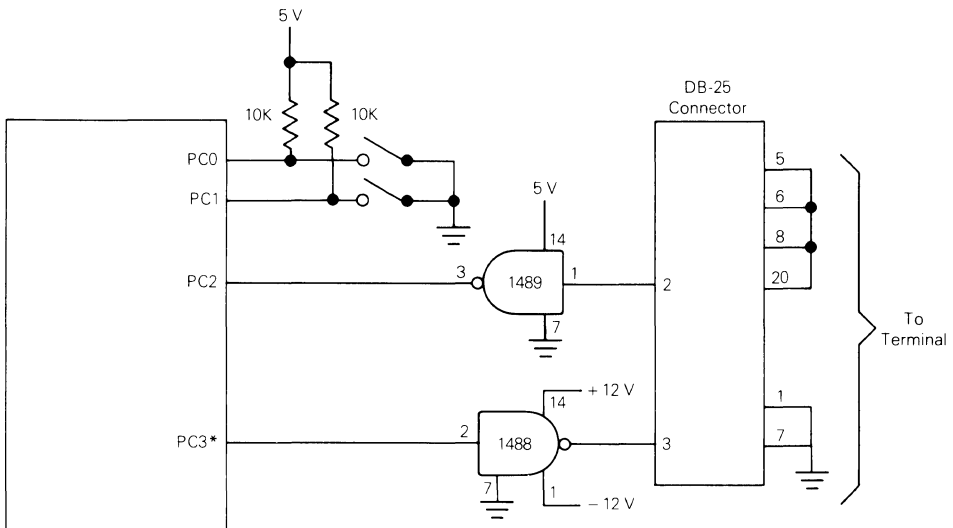
Software Applications

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 CDP6805 CMOS Family software features and include some commonly used routines.

Serial I/O Software For RS-232

The example discussed here uses two I/O port lines for the serial input and output lines. Figure 14 contains a schematic diagram of an RS-232 interface for serial I/O. Included as part of Figure 14 is the baud rate selection table showing baud rates of 300, 1200, 4800 and 9600. The example subroutine is illustrated in Figure 15. 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 lines commented by "CMOS DITTO" or "CMOS EQUALIZATION" are necessary to "make-up" for the generally fewer cycles-per-instruction of the CDP6805 CMOS Family members when compared with the M6805 HMOS devices.



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 14 - 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 15 - 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 15 - Serial I/O Software Subroutine Example (Continued)

Keypad Scan Routine

A common task for control-oriented microprocessors is to scan a 4 x 4 keypad, such as the one illustrated in the example of Figure 16. The example shown uses port A lines 4-7 as scanning outputs and port A lines 0-3 as sensing inputs. It is often desirable to place CDP6805 CMOS Family members in a low-power mode; therefore, the STOP instruction is incorporated in the routine shown in Figure 17.

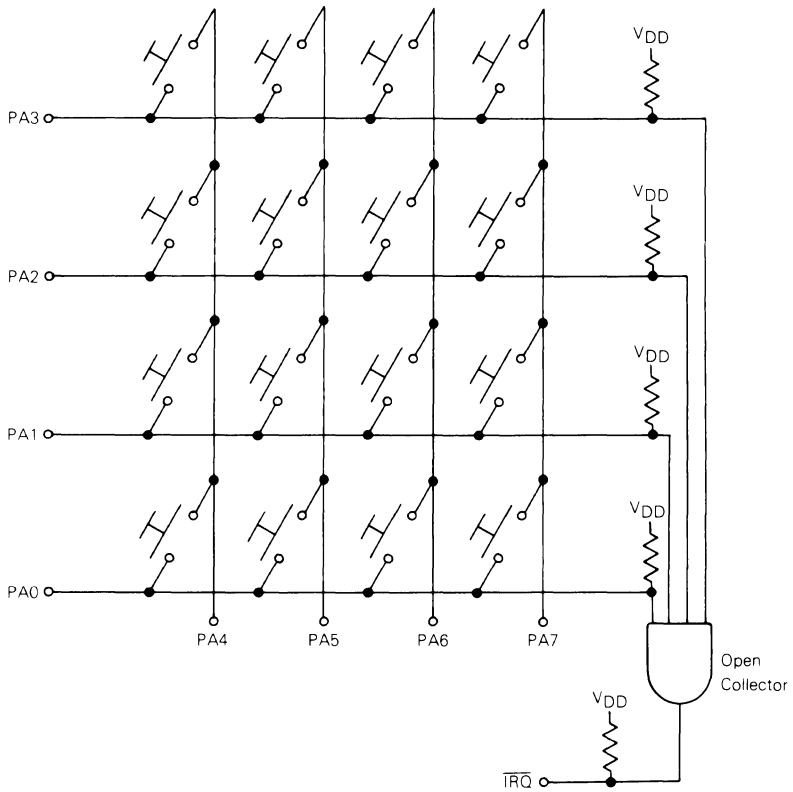


Figure 16 - 4 x 4 Keypad and Closure Detection Circuit Schematic Diagram

The example shown in Figure 17 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 $\overline{IRQ}$ goes low the KEYSCN vector is selected and calls the KEYSCN interrupt service routine. The interrupt service routine first causes $\overline{IRQ}$ 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. Debounce consists of checking for a keypad switch closure after a 1536 bus cycle delay to assure that the interrupt was not the result of noise. 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. 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 switch was closed. The 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 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.

```
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 17. KEYSCN Routine Example

The CDP68HC05D2 features software programmable open drain outputs on PORT A and a PORT B interrupt which are helpful in the design of a keyboard interface.

Stack Handling

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 stored 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.

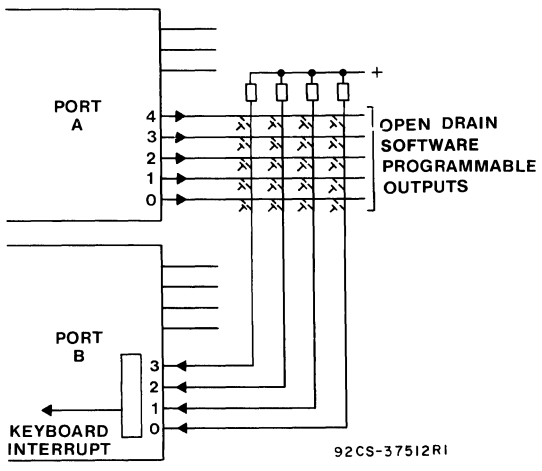


Figure 18 - Alternative Keyboard Interface Utilizing CDP68HC05D2

The CDP6805 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 19 can provide the CDP6805E2 MPU with additional stack capability for temporary variable storage. In this routine, a temporary location called POINTR serves to hold the relative address of the next free stack location. When the routine is entered, the contents of POINTR 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. Because 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 19 uses two fixed temporary locations: one (called POINTR) 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 CDP6805E2, 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 19 - Stack Emulation Routine

Block Move

This example makes a copy of a block of data in another portion of memory. The indexed addressing modes of the CDP6805 CMOS Family make a block move relatively simple.

The routine is shown in Figure 20. 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).

```
                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 20 - Block Move Routine Example

DAA (Decimal Adjust Accumulator)

Although the CDP6805 CMOS Family is primarily a controller, it is occasionally required to perform arithmetic operations on BCD numbers. Because 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 21. 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                          *
00005                          *
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 21 - DAA Subroutine Example

Multiply

Multiply subroutines for either 16-bit x 16-bit or 8-bit x 8-bit multiplications can be written using less than 30 bytes. Examples of both cases are illustrated in Figures 22 and 23. Note that the CDP68HC05C4 and CDP68HC05D2 have a multiply instruction.

```
PAGE  001  DPMUL05 .SA:0

00001                          *
00002                          *
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 22 - 16-bit x 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 23 - 8-bit x 8-bit Multiplication Subroutine Example

Divide

Two examples of subroutines which can be used for performing division of two numbers are illustrated in Figures 24 and 25. 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.

```
                    ********************************************************************
                    *
                    *       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.
                    *
                    *
                    *
                    *       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 24 - 16-bit ÷ 16-bit With 8-bit Result Subroutine Example

```
                        *
        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 24 - 16-bit ÷ 16-bit With 8-bit Result Subroutine Example (Continued)

```
          DIV05   .SA:0

00001                       *
00002                       *
00003                       *  16 BIT UNSIGNED DIVIDE (16 BIT RESULT)
00004                       *  16 BIT DIVIDEND IN DVSOR & DVSOR+1
00005                       *  16 BIT DIVISOR IN DVDND & DVDND+1
00006                       *  16 BIT RESULT IN RESLT & RESLT+1
00007                       *
00008A 0064                        ORG     $64
00009                       *
00010A 0064    0001      A COUNT   RMB     1
00011A 0065    0002      A DVSOR   RMB     2
00012A 0067    0002      A DVDND   RMB     2
00013A 0069    0002      A RESLT   RMB     2
00014A 006B    0001      A TEMPA   RMB     1
00015A 006C    0001      A TEMPX   RMB     1
00016                       *
00017A 0080                        ORG     $80
00018A 0080 A6 01         A STRT    LDA     #1
00019A 0082 3D 65         A         TST     DVSOR
00020A 0084 2B 0B      0091         BMI     DIV153
00021A 0086 4C                DIV151 INCA
00022A 0087 38 66         A         ASL     DVSOR+1
00023A 0089 39 65         A         ROL     DVSOR
00024A 008B 2B 04      0091         BMI     DIV153
00025A 008D A1 11         A         CMP     #17
00026A 008F 26 F5      0086         BNE     DIV151
00027A 0091 B7 64         A DIV153  STA     COUNT
00028A 0093 B6 67         A         LDA     DVDND
00029A 0095 BE 68         A         LDX     DVDND+1
00030A 0097 3F 67         A         CLR     DVDND
00031A 0099 3F 68         A         CLR     DVDND+1
00032A 009B BF 6C         A DIV163  STX     TEMPX
00033A 009D B7 6B         A         STA     TEMPA
00034A 009F 9F                      TXA
00035A 00A0 B0 66         A         SUB     DVSOR+1
00036A 00A2 B7 6C         A         STA     TEMPX
00037A 00A4 B6 6B         A         LDA     TEMPA
00038A 00A6 B2 65         A         SBC     DVSOR
00039A 00A8 B7 6B         A         STA     TEMPA
00040A 00AA BE 6C         A         LDX     TEMPX
00041A 00AC 24 0E      00BC         BCC     DIV165
00042A 00AE 9F                      TXA
00043A 00AF BB 66         A         ADD     DVSOR+1
00044A 00B1 B7 6C         A         STA     TEMPX
00045A 00B3 B6 6B         A         LDA     TEMPA
00046A 00B5 B9 65         A         ADC     DVSOR
00047A 00B7 B7 6B         A         STA     TEMPA
00048A 00B9 98                      CLC
00049A 00BA 20 01      00BD         BRA     DIV167
00050A 00BC 99                DIV165 SEC
00051A 00BD 39 68         A DIV167  ROL     DVDND+1
00052A 00BF 39 67         A         ROL     DVDND
00053A 00C1 34 65         A         LSR     DVSOR
00054A 00C3 36 66         A         ROR     DVSOR+1
00055A 00C5 3A 64         A         DEC     COUNT
00056A 00C7 26 D2      009B         BNE     DIV163
00057A 00C9 B6 67         A         LDA     DVDND
00058A 00CB B7 69         A         STA     RESLT
          DIV05   .SA:0

00059A 00CD B6 68         A         LDA     DVDND+1
00060A 00CF B7 6A         A         STA     RESLT+1
00061A 00D1 81                      RTS
00062                       *
00063                              END
```

Figure 25 - 16-bit ÷ 16-bit With 16-bit Result Subroutine Example

Hardware Features

Introduction

Each member of the CDP6805 CMOS Family (except for the CDP6805E2 and CDP6805E3) contains, on-chip, nearly all of the support hardware necessary for a complete processor system. The block diagram of Figure 26 shows a central processing unit (CPU) which is identical for all members of the family, including the CDP6805E2 and CDP6805E3. There is one main difference in various family members and that is the size of the stack pointer and program counter registers. Because the size of these two registers is determined by the amount of device memory, they vary from 11 bits to 16 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 CDP6805E2 and CDP6805E3) 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, (s)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 or an external bus.

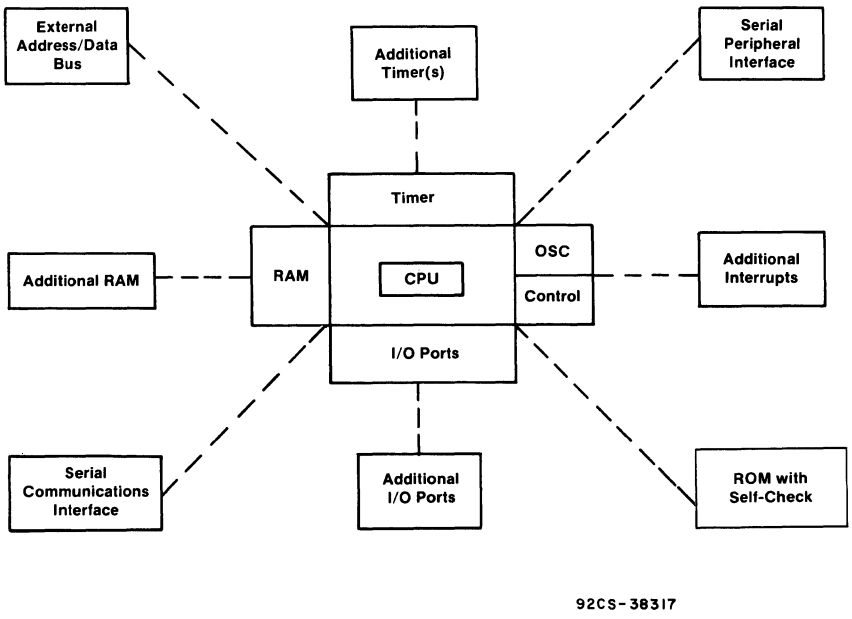


Figure 26 - CDP6805 CMOS Family Block Diagram

The CDP6805 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, like the timer, 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 CDP6805 CMOS Family I/O features is in learning how the peripheral registers effect the device operation. Because 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.

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, because RAM can be read from or written to, it is best used for storing variables. All on-chip RAM is contained in the first 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 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.

Permanent Storage (ROM)

CMOS Family devices, except the CDP6805E2 and CDP6805E3, contain some permanent, non-volatile mask-programmed read-only memory (ROM). Non-volatile memory is generally used to store the user programs as well as tables and constants.

Oscillator

The on-chip oscillator contained on every CDP6805 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 27. Each mode has its advantages and the basic trade-off is between economy and accuracy. An external CMOS oscillator is recommended when a crystal outside the specified range of the part is to be used.

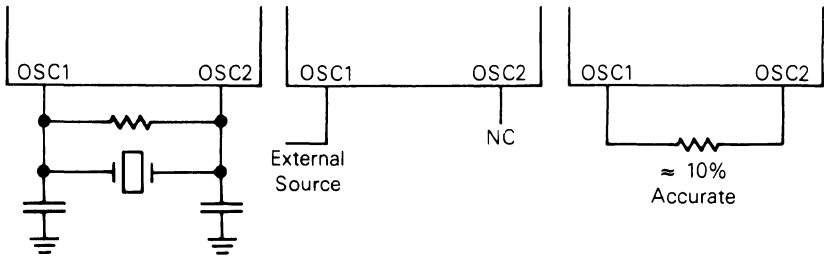


Figure 27 - CDP6805 CMOS Family Oscillator Modes

The CDP6805E2 and CDP6805E3 provide for only two types of oscillator inputs — crystal circuit or external clock. The CDP6805 CMOS Family MCU's (CDP6805F2/G2 and CDP68HC05C4/D2) use a manufacturing mask option to select either the crystal or resistor circuit. A second manufacturing mask option provides either a divide-by-two or divide-by-four circuit to produce the internal system clock. An external clock may be used with either the RC or crystal oscillator mask option.

Resets

The CDP6805 CMOS Family processor can be reset in two ways: either by initial power-up or by the external reset input pin ($\overline{\text{RESET}}$) (refer to Figure 27). Any of the reset methods allow an orderly start-up. Additionally, the $\overline{\text{RESET}}$ input can be used to exit the STOP and WAIT modes of program execution.

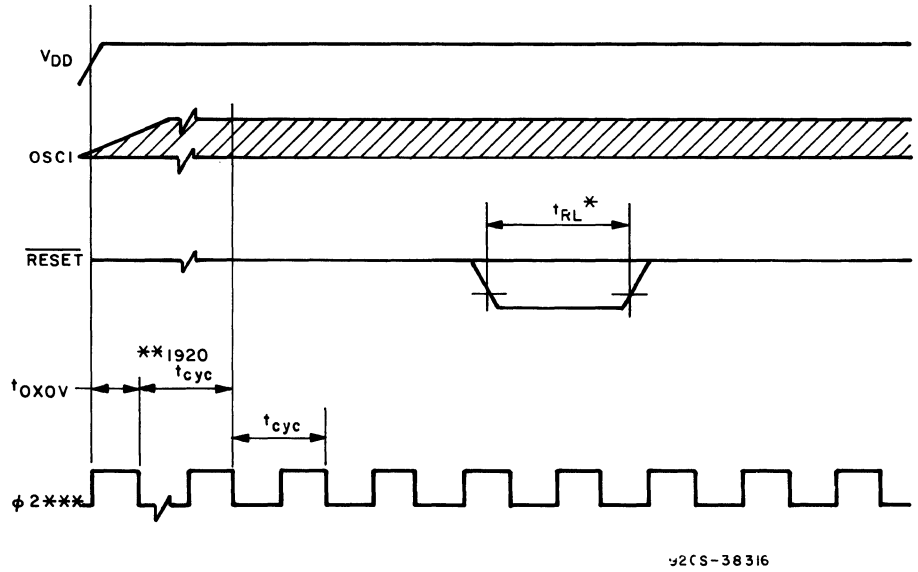
The external $\overline{\text{RESET}}$ input allows the processor to recover from otherwise catastrophic errors. When using the external reset mode, the $\overline{\text{RESET}}$ pin must stay low for some minimum time as shown in Fig. 28. External reset ($\overline{\text{RESET}}$) is implemented with a Schmitt trigger input for improved noise immunity.

The power-on reset occurs when a positive transition is detected on V_{DD}. 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 storage. There is no provision for a power-down reset. The power-on circuitry provides for a delay from the time of first oscillator operation. If the external $\overline{\text{RESET}}$ pin is low at the end of the time out, the processor remains in the reset condition until the $\overline{\text{RESET}}$ pin goes high.

Any reset causes the following to occur:

1. All interrupt requests are cleared to "0".
2. All interrupt enables in the peripheral control registers are cleared to disable interrupts.

3. The condition code register interrupt mask bit (I) is set to a "1" (disabled).
4. All data direction registers are cleared to "0" (input).
5. The stack pointer is reset to the top of stack.
6. The address bus is forced to the reset vector. (The reset vector contains the address of the reset routine.)
7. The STOP and WAIT latches are reset.
8. The external interrupt latch is reset.



*The $\overline{\text{RESET}}$ pulse width is a minimum of one t_{cyc} for all members of the CDP6805 CMOS Family except the CDP68HC05C4 and CDP68HC05D2 which require the $\overline{\text{RESET}}$ pin to stay low for a minimum of one and one half t_{cyc}.

**The delay from the time that the oscillator becomes active is 4064 t_{cyc} in the CDP68HC05C4 and CDP68HC05D2, when programmed for a crystal oscillator. When programmed for an RC oscillator the delay is 2 t_{cyc} for the CDP68HC05D2.

***Internal timing signal not available externally.

Figure 28 - Power-on Reset and $\overline{\text{RESET}}$

Interrupts

Systems often require that normal processing be interrupted so that some other event may be serviced. The CMOS Family program execution may be interrupted in the following ways:

1. Externally via the $\overline{\text{IRQ}}$ pin. External interrupts are maskable.
2. Internally with the on-chip timer. The timer interrupt is maskable.
3. Internally with the serial communications interface on the CDP68HC05C4. The serial communications interface interrupts are maskable.
4. Internally with the serial peripheral interface provided on the CDP68HC05C4 and CDP68HC05D2. The SPI system interrupts are maskable.
5. Internally with the Port B interrupt on the CDP68HC05D2. The Port B interrupt is maskable.
6. Internally by executing the software interrupt instruction (SWI). The SWI is non-maskable.

Interrupts such as timer, SPI, and SCI have several flags which will cause the interrupt. Generally interrupt flags are located in associated control registers. The interrupt flags and enable bits are never contained in the same register (except for the Port B interrupt). If the enable bit is a logic zero, it blocks the interrupt from occurring but does not inhibit the flag from being set. The general sequence for clearing interrupt is a software sequence of first accessing the status register while the interrupt flag is set, followed by a read or write of an associated register. Reset clears all enable bits to preclude interrupts during the reset procedure.

When an external timer, SCI, SPI, or Port B 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. When an unmasked interrupt is recognized, the current state of

the machine is pushed onto the stack, the interrupt mask bit in the condition code register is set to prevent further interrupts, the program counter is loaded with the address of the interrupt service routine, and the interrupt service routine is executed. If both an external and an internal hardware interrupt are pending, the external interrupt is serviced first; however, the internal hardware 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 ($\overline{IRQ}$) may be tested with the BIL or BIH conditional branch instructions. These instructions may be used to allow the external interrupt pin to be used as an additional input pin regardless of the state of the interrupt mask in the condition code register.

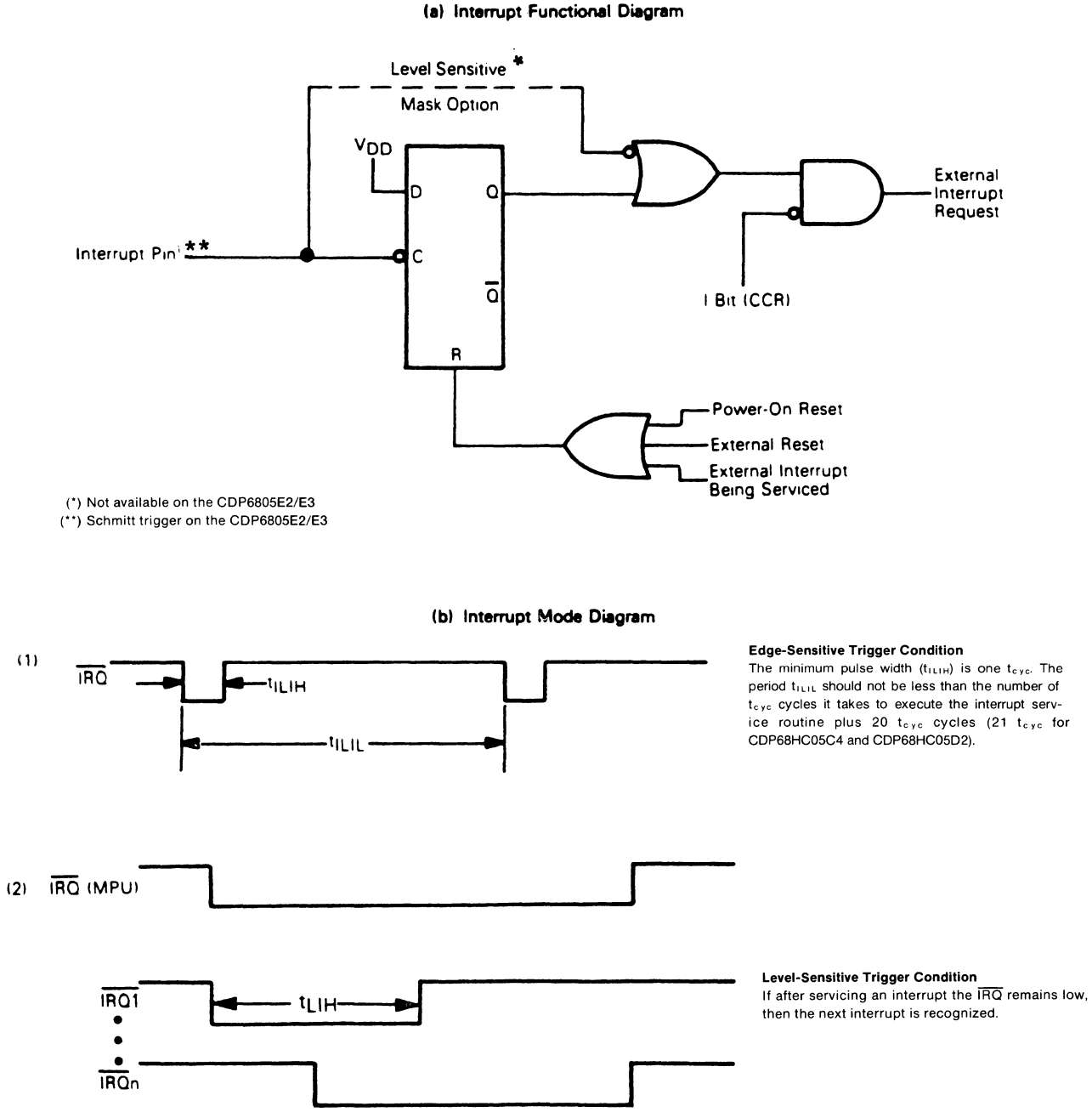


Figure 29 — External interrupt.

External Interrupts

All external interrupts are maskable. If the interrupt mask bit (I) of the condition code register is set, all interrupts are disabled. Clearing the I bit enables the external interrupts. If the interrupt mask bit of the condition code register is cleared and the external interrupt pin $\overline{IRQ}$ is "low" or a negative edge has set the internal interrupt flip-flop, then the external interrupt occurs. When the interrupt is recognized, the current state of the machine is pushed onto the stack, the interrupt mask bit in the condition code register is set to prevent further interrupts until this one is serviced, the program counter is loaded with the appropriate vector location contents and the interrupt routine is executed. Either a level- and edge-sensitive or edge-sensitive only input is available as a mask option on the CDP6805F2/G2 and CDP68HC05C4/D2. Figure 29 shows both a functional diagram and timing for the interrupt line. The timing diagram shows two different treatments of the interrupt line ($\overline{IRQ}$) to the processor. The first method shows single pulses on the interrupt line spaced far enough apart to be serviced. The minimum time between pulses is a function of the length of the interrupt service routine. Once a pulse occurs, the next pulse should not occur until the MPU software has exited the service routine (an RTI occurs). This time (t_{ILIL}) is obtained by adding 20 instruction cycles (t_{cyc}) (21 cycles for the CDP68HC05C4 and CDP68HC05D2) to the total number of cycles it takes to complete the service routine including the RTI instruction; refer to Figure 29. The second configuration shows many interrupt lines "wire ORed" to form the interrupts at the processor. Thus, if after servicing one interrupt the interrupt line remains low, then the next interrupt is recognized.

Timer Interrupt CDP6805E2/E3/F2/G2

Each time the timer on the CDP6805E2/E3/F2/G2 decrements to zero (transitions from $01 to $00), the timer interrupt request bit (TCR7) is set. The processor is interrupted only if the timer mask bit (TCR6) and interrupt mask bit (I bit) are both cleared. When the interrupt is recognized, the current state of the machine is pushed onto the stack and the interrupt mask bit (I) in the condition code register is set. This mask prevents further interrupts until the present one is serviced. The processor now vectors to the timer service routine by loading the program counter with the contents of the timer interrupt vector. Notice that if the timer is being used to exit the WAIT mode, 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.

Timer Interrupts CDP68HC05C4/D2

There are three different timer interrupt flags that will cause a timer interrupt whenever they are set and enabled. These three interrupt flags are found in the three most significant bits of the timer status register (TSR). All three timer interrupts will vector to the same service routine location. The three timer interrupt conditions are timer overflow, output compare, and input capture. All interrupt flags have corresponding enable bits (ICIE, OCIE, TOIE) found in the timer control register (TCR). Reset clears all enable bits, thus preventing an interrupt from occurring during the reset time period. The actual processor interrupt is generated only if the interrupt mask bit (I) in 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 interrupt mask bit in the condition code register is set. This masks further interrupts until the present one is serviced. The general sequence for clearing an interrupt is a software sequence of accessing the status register while the flag is set, followed by a read or write of an associated register. Further information can be found in the Timer description.

Serial Communications Interface (SCI) Interrupts CDP68HC05C4

An interrupt in the serial communications interface (SCI) of the CDP68HC05C4 occurs when one of the interrupt flag bits in the serial communications status register is set, provided the I bit in the condition code register is clear and the enable bit in the serial communications control register is enabled. When the interrupt is recognized, the current state of the machine is pushed onto the stack, the I bit in the condition code register is set to prevent further interrupts until the present one is serviced, and the program counter vectors to the serial interrupt service routine. Software in the serial interrupt service routine must determine the priority and cause of the SCI interrupt by examining the interrupt flags and status bits located in the serial communications register. The general sequence for clearing an interrupt is a software sequence of accessing the serial communications status register while the flag is set followed by a read or write of an associated register. Refer to the section on the Serial Communications Interface for a description of the SCI system and its interrupts.

Serial Peripheral Interface (SPI) System Interrupts CDP68HC05C4/D2

An interrupt in the serial peripheral interface (SPI) of the CDP68HC05C4 and CDP68HC05D2 occurs when one of the interrupt flag bits in the serial peripheral status register is set, provided the interrupt mask bit (I) of the condition code register is clear and the enable bit in the serial peripheral control reg-

ister is enabled. When the interrupt is recognized, the current state of the machine is pushed onto the stack, the interrupt mask bit in the condition code register is set to mask further interrupts until the present one is serviced, and the program counter vectors to the SPI interrupt service routine. Software in the serial peripheral interface service routine must determine the priority and cause of the SPI interrupt by examining the interrupt flag bits located in the SPI status register. The general sequence for clearing an interrupt is a software sequence of accessing the status register while the flag is set, followed by a read or write of an associated register. Refer to the section on the Serial Peripheral Interface for a description of the SPI system and its interrupts.

Port B Interrupt CDP68HC05D2

A Port B interrupt on the CDP68HC05D2 will occur when any one of the eight port lines (PB0-PB7) is pulled to a low level, provided the interrupt mask bit of the condition code register is clear and the enable bit in the miscellaneous control register is enabled. Before enabling Port B interrupts, PB0 through PB7 should be programmed as inputs, i.e., their corresponding DDR bits must be zero. Programming any of these lines as outputs inhibits them from generating an interrupt.

The purpose of this interrupt is to provide easy use of the PB0-PB7 lines as keyboard sense inputs. For systems where the keyboard response is not interrupt driven, this interrupt can be disabled. Port B interrupts will also cause an exit from the stop mode.

Software Interrupt CDP6805 Family

The software interrupt is executed the same as any other instruction and as such will take precedence over hardware interrupts only if the I bit is set (interrupts masked). The SWI is executed regardless of the state of the interrupt mask in the condition code register; however, when the I bit is clear (interrupts enabled) 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. Recall however that the hardware interrupts wait for the current instruction to complete, including the software interrupt instruction, before they are serviced. The execution of the SWI instruction is similar to hardware interrupts in that the I bit is set, CPU registers are stacked, etc. The SWI uses its own unique vector location.

Stop

The STOP instruction places the CDP6805 in its lowest power consumption mode. In the STOP function, the internal oscillator is turned off causing all internal processing and the timer to be halted; refer to Figure 30. In the CDP6805E2/E3/F2/G2 the timer control register (TCR) bits 6 and 7 are altered to remove any pending timer interrupt requests and to disable any further timing interrupts. External interrupts are enabled in the condition code register (I = 0). All other registers, memory, and I/O lines remain unaltered. The processor can be brought out of the STOP mode by an external $\overline{IRQ}$, $\overline{RESET}$, a Port B interrupt, or a timer interrupt if the timer oscillator (as opposed to the internal oscillator) is being used.

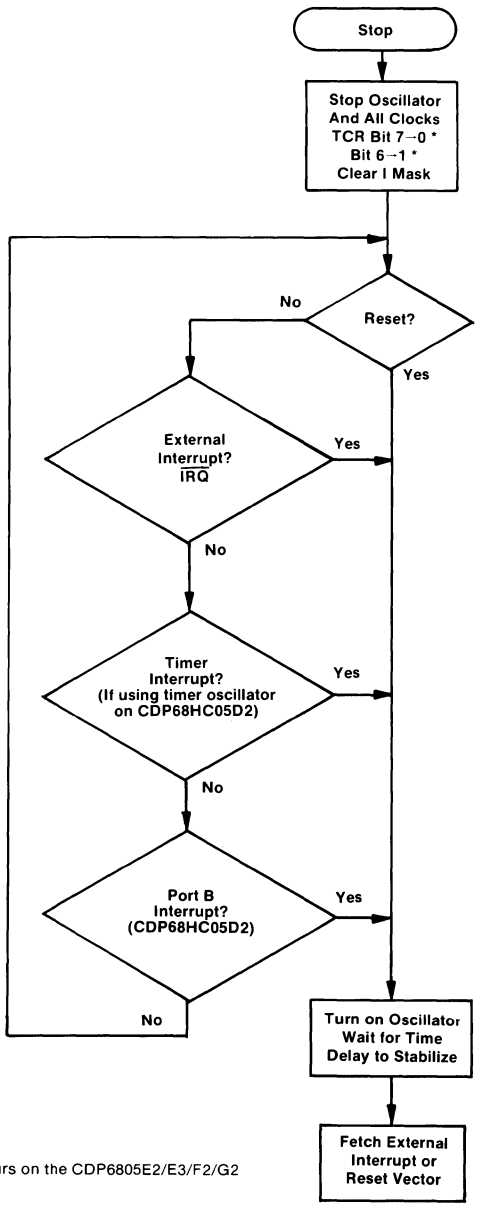


Figure 30 - STOP Function Flowchart

Wait

The WAIT instruction places the CDP6805 in a low-power consumption mode. The WAIT mode consumes somewhat more power than the STOP mode. In the WAIT mode, the internal clock remains active, and all CPU processing is stopped; however, the programmable timer, serial peripheral interface, and serial communications interface systems remain active. Refer to Figure 31. During the WAIT mode, the I bit in the condition code register is cleared to enable interrupts. All other registers, memory, and I/O lines remain in their last state. An interrupt or reset brings the processor out of the WAIT mode. The timer may be enabled by software prior to entering the WAIT mode to allow a periodic exit from the WAIT mode. If an external and a timer interrupt occur at the same time, the external interrupt is serviced first; then, if the timer interrupt request is not cleared in the external interrupt routine, the normal timer interrupt (not the timer WAIT interrupt) is serviced since the MCU is no longer in the WAIT mode.

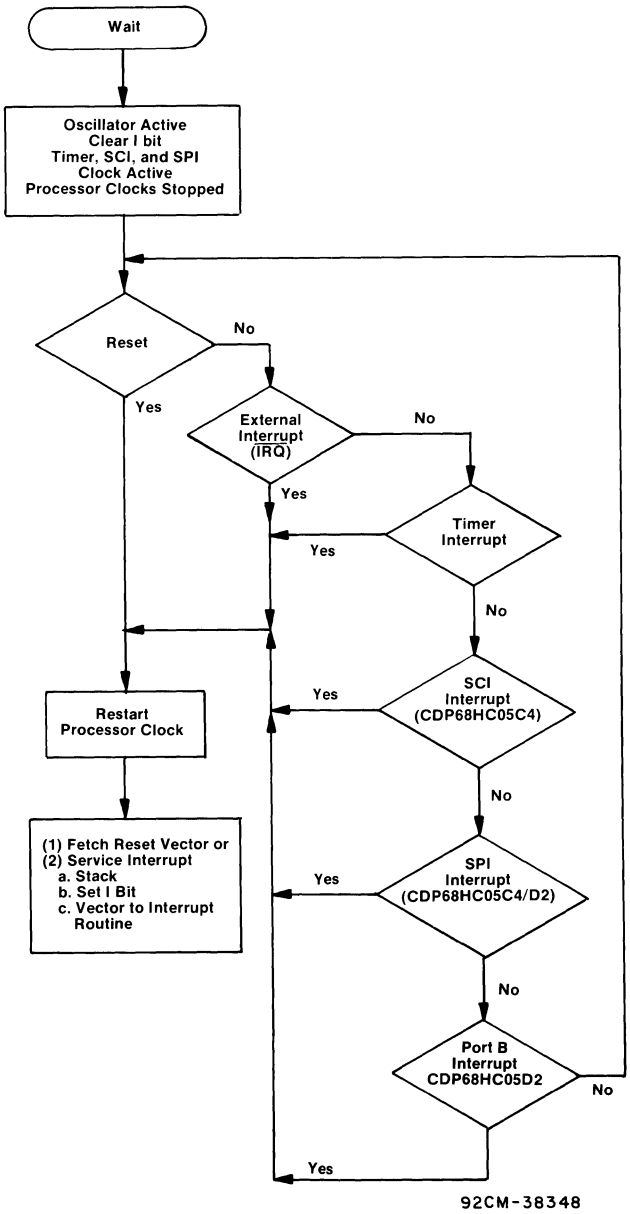


Figure 31 - WAIT Function Flowchart

I/O Ports

At least 13 individually programmable, bidirectional I/O lines are included on each member of the CDP6805 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 32. A pin is configured as an output if its corresponding DDR bit is set to a logic "1". A pin is configured as an input if its corresponding DDR bit is cleared to a logic "0". At reset, all DDR's are cleared, which configures all port pins as inputs. The data direction registers are able to be written to or read by the processor and may be used with RMW instructions. 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. When programmed as an input, the input data bit(s) are not latched. An MPU read of the port bits programmed as outputs reflects the last value written to that location. An MPU read of the port bits programmed as inputs reflects the current status of the corresponding input pins. Table II provides a description of the effects of port data register operation.

The CDP68HC05C4/D2 includes a number of input-only lines. These lines have no DDR and have read-only data registers.

PORTS A, B and C on the CDP68HC05C4 may be programmed as inputs or outputs. Port D on the CDP68HC05C4 is a 7-bit fixed input port (PD0-PD5, PD7) that continually monitors the external pins whenever the SPI or SCI systems are disabled. During power-on reset or external reset all seven bits become valid input ports because all special function output drivers are disabled. For example, with the serial communications interface (SCI) system enabled (RE = TE = 1), PD0 and PD1 inputs will read zero. With the serial peripheral interface (SPI) system disabled (SPE = 0), PD2 through PD5 will read the state of the pin at the time of the read operation. No data register is associated with the port when it is used as an input.

PORTS A, B and C on the CDP68HC05D2 may be programmed as inputs or outputs. PORT A can be programmed to be open-drain outputs when bit 0 in the Miscellaneous Control/Status Register is set

and their DDR bits are set. PORT B features an interrupt which occurs when any one of the eight port lines is pulled low provided the registers are properly enabled.

PORT D contains four individually programmable bidirectional lines and three input lines. The direction of the four bidirectional lines is determined by the state of the data direction register (DDR). Each of these four lines has an associated DDR bit. The validity of a port bit is determined by whether the SPI system and external timer oscillator are enabled or disabled. If the SPI is enabled and bit 5 of the SPI control register is true, PORT D bits 2-5 become open-drain outputs. An exception is PD6, the timer output compare pin (TCMP), which is always an output. Upon power-on-reset or external reset all bits, except PD0 (output pin for the timer oscillator) and PD6, contain input port data because all other special function output drivers are disabled.

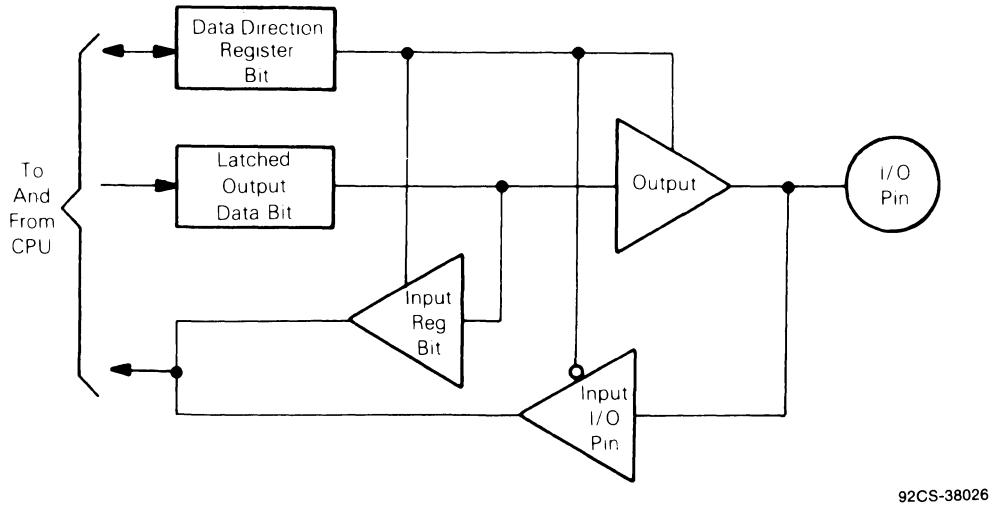


Figure 32 - Typical Port I/O Circuitry

TABLE II. Port Data Register Accesses

R/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/W is an internal line.

Timer Description

All CDP6805 CMOS Family devices contain a timer on chip.

Timer CDP6805E2/E3/F2/G2

The timer on the CDP6805E2/E3/F2/G2 contains an 8-bit software programmable counter with a 7-bit software selectable prescaler. Figure 33 contains a block diagram of the timer. The counter may be preset under program control and decrements towards 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 occurs before the timer interrupt is serviced, the interrupt is lost. TCR7 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 within the range of ÷ 1 to ÷ 128 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 one. This allows for truncation-free counting.

The timer input can be configured for 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 section.

Timer Input Mode 1

If TCR5 and TCR4 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.

Timer Input Mode 2

With TCR5=0 and TCR4=1, 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 timer input pulse simply turns on the internal clock for the duration of the pulse. The resolution of the count in this mode is ± one internal clock and, therefore, accuracy improves with longer input pulse widths.

Timer Input Mode 3

If TCR5=1 and TCR4=0, all inputs to the timer are disabled.

Timer Input Mode 4

If TCR5=1 and TCR4=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 on the falling edge of the external signal.

Figure 33 shows a block diagram of the timer subsystem. Power-on reset and the STOP instruction invalidate the contents of the counter.

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 $\overline{\text{RESET}}$, 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 $\overline{\text{RESET}}$, 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 $\overline{\text{RESET}}$.)
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 $\overline{\text{RESET}}$.)
1 — Enable external TIMER pin.
0 — Disable external TIMER pin.

TCR5	TCR4	
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 33 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 $\overline{\text{RESET}}$.)

TCR2, TCR1, TCR0 — Prescaler select bits: decoded to select one of eight outputs on the prescaler. (Unaffected by $\overline{\text{RESET}}$.)

	Prescaler		
TCR2	TCR1	TCR0	Result
0	0	0	÷1
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

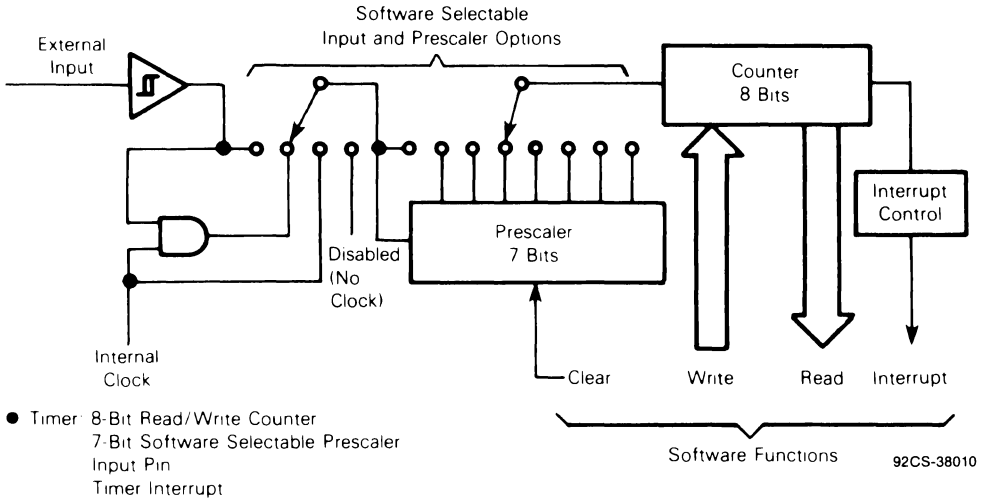


Figure 33 - Programmable Timer/Counter Block Diagram.

Programmable Timer CDP68HC05C4/D2

A 16-bit programmable timer, preceded by a fixed prescaler, is available on the CDP68HC05C4 and CDP68HC05D2. It can be used for many purposes including measuring the pulse width of an input signal while simultaneously generating an output signal. Pulse widths for both input and output signals can vary from several microseconds to many seconds. The timer is also capable of generating periodic interrupts, indicating passage of an arbitrary number of MCU cycles or counts from an external oscillator. A block diagram of the timer is shown in Figure 34.

Because the timer has a 16-bit architecture, each specific functional segment is represented by two registers. These registers contain the high and low byte of that functional segment. Generally, accessing the low byte of a specific timer function allows full control of that function; however, an access of the high byte inhibits that specific timer function until the low byte is also accessed.

Note

The I bit in the condition code register should be set while manipulating both the high and low byte register of a specific timer function to ensure that an interrupt does not occur. A problem could arise if an interrupt occurred in the interval of time between the access of the high and low byte.

A description of the timer registers follows:

Counter Register

The key element in the programmable timer is a 16-bit free-running counter, or counter register, preceded by a prescaler which divides the internal processor clock by four. The prescaler gives the timer a resolution of 2.0 microseconds if the internal processor clock is 2.0 MHz. The counter is clocked to increasing values during the low portion of the internal processor clock. Software can read the counter at any time without affecting its value.

The double byte free-running counter can be read from either of two locations. One double byte is called the counter register, and the other double byte is called the counter alternate register. A read sequence containing only a read of the least significant byte of the free-running counter, from either double byte, will receive the count value at the time of the read. If a read of the free-running counter or counter alternate register first addresses the most significant byte, it causes the least significant byte to be transferred to a buffer. This buffer value remains fixed after the first most significant byte read even if the user reads the most significant byte several times. This buffer is accessed when reading the free-running counter or counter alternate register's least significant byte, after a read of the most significant byte, and thus completes a read sequence of the total counter value. Notice that in reading either the free-running counter or counter alternate register, if the most significant byte is read, the least significant byte must also be read in order to complete the sequence.

The free-running counter is set to some constant during reset and is always a read-only register. During a power-on reset, the counter is also configured to the constant and begins running after the oscillator startup delay. Because the free-running counter is 16 bits preceded by a fixed divide-by-four prescaler, the value in the free-running counter repeats every 262,144 MCU internal processor clock cycles. When the counter rolls over from $FFFF to $0000, the timer overflow flag (TOF) bit is set. An interrupt can also be enabled when counter rollover occurs by setting its interrupt enable bit (TOIE).

Output Compare Register

The output compare register is a 16-bit read/write

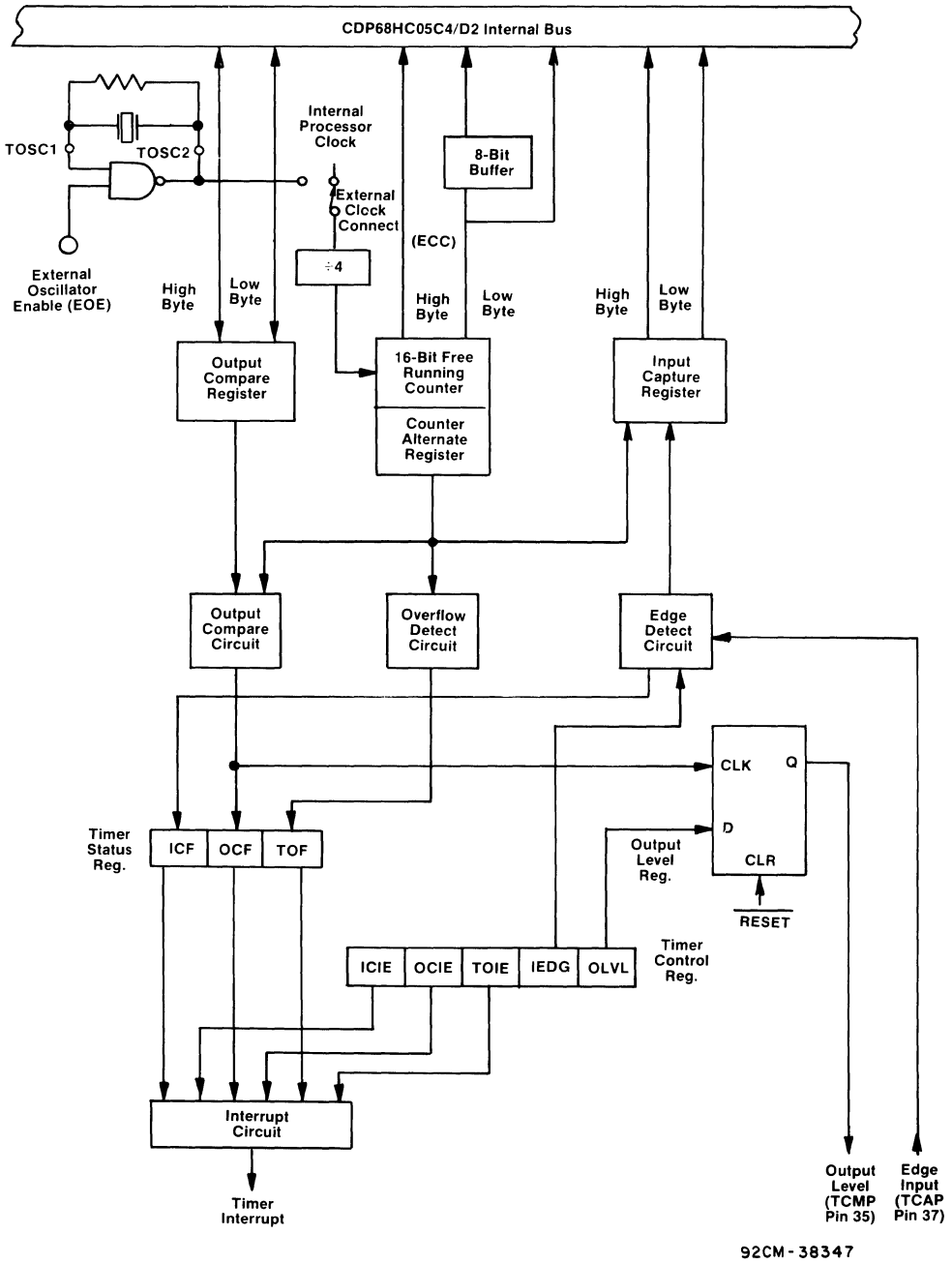


Figure 34 - Programmable Timer Block Diagram

*Timer clock source option available on the CDP68HC05D2.

register which is made up of two 8-bit registers. The output compare register can be used for several purposes such as controlling an output waveform or indicating when a period of time has elapsed. The output compare register is unique in that all bits are readable and writable and are not altered by the timer hardware. Reset does not affect the contents of this register. If the compare function is not utilized, the two bytes of the output compare register can be used as storage locations.

The contents of the output compare register are compared with the contents of the free-running counter once during every four internal processor clocks. If a match is found, the corresponding output compare flag (OCF) is set and the corresponding output level (OLVL) bit is clocked (by the output compare circuit pulse) to an output level register. The values in the output compare register and the output level bit should be changed after each successful comparison in order to control an output

waveform or establish a new elapsed timeout. An interrupt can also accompany a successful output compare provided that the corresponding interrupt enable bit, OCIE, is set.

After a processor write cycle to the output compare register containing the most significant byte, the output compare function is inhibited until the least significant byte is also written. The user must write both locations if the most significant byte is written first. A write made only to the least significant byte will not inhibit the compare function. The free-running counter is updated every four internal processor clock cycles due to the internal prescaler. The minimum time required to update the output compare register is a function of the software program rather than the internal hardware.

A processor write may be made to either byte of the output compare register without affecting the other byte. The output level (OLVL) bit is clocked to the output level register regardless of whether the output compare flag (OCF) is set or clear.

Because neither the output compare flag (OCF bit) nor the output compare register is affected by reset, care must be exercised when initializing the output compare function with software. The following procedure is recommended:

1. Write the high byte of the output compare register to inhibit further compares until the low byte is written.
2. Read the timer status register to arm the OCF if it is already set.
3. Write the output compare register low byte to enable the output compare function with the flag clear.

The advantage of this procedure is to prevent the OCF bit from being set between the time it is read and the write to the output compare register. A software example is shown below.

```
B7  16  STA  OCMPHI   Inhibit Output Compare
B6  13  LDA  TSTAT    Arm OCF Bit if Set
BF  17  STX  OCMPLD   Ready for Next Compare
```

Input Capture Register

The two 8-bit registers which make up the 16-bit input capture register are read-only and are used to latch the value of the free-running counter after a defined transition is sensed by the corresponding input capture edge detector. The level transition which triggers the counter transfer is defined by the corresponding input edge bit (IEDG). Reset does not affect the contents of the input capture register.

The result obtained by an input capture will be one more than the value of the free-running counter on the rising edge of the internal processor clock preceding the external transition. This delay is required for internal synchronization. Resolution is affected by the prescaler allowing the timer to only increment every four internal processor clock cycles.

The free-running counter contents are transferred to the input capture register on each proper signal transition regardless of whether the input capture flag (ICF) is set or clear. The input capture register always contains the free-running counter value which corresponds to the most recent input capture.

After a read of the most significant byte of the input capture register, counter transfer is inhibited until the least significant byte of the input capture register is also read. This characteristic forces the minimum pulse period attainable to be determined by the time used in the capture software routine and its interaction with the main program. A polling routine using instructions such as BRSET, BRA, LDA, STA, INCX, CMBX, and BEG might take 34 machine cycles to complete. The free-running counter increments every four internal processor clock cycles due to the prescaler.

A read of the least significant byte of the input capture register does not inhibit the free-running counter transfer. Again, minimum pulse periods are ones which allow software to read the least significant byte and perform needed operations. There is no conflict between the read of the input capture register and the free-running counter transfer since they occur on opposite edges of the internal processor clock.

Timer Control Register (TCR)

The timer control register (TCR) is an 8-bit read/write register which contains control bits. The timer control register on the CDP68HC05C4 contains five control bits; the timer control register on the CDP68HC05D2 contains an additional two control bits. Three of the five bits in common between the CDP68HC05C4 and CDP68HC05D2 control interrupts associated with each of the three flag bits found in the timer status register (discussed below). The other two bits control: 1) which edge (negative or positive) is significant to the input capture edge detector and 2) the next value to be clocked to the output level register in response to a successful output compare. The two extra bits found in the CDP68HC05D2 control: 1) the source of the timer clock and 2) whether the external timer oscillator is enabled. The timer control register and the free-running counter are the only sections of the timer affected by rest. The TCMP pin is forced low during external reset and stays low until a valid compare changes it to a high. The timer control register is illustrated below followed by a definition of each bit.

7	6	5	4*	3*	2	1	0
ICIE	OCIE	TOIE	EOE	ECC	—	IEDG	OLVL

*Available only on the CDP68HC05D2.

Bit 7 ICIE Input Capture Interrupt Enable
If set, a timer interrupt is enabled whenever the ICF status flag is set; if clear, the interrupt is inhibited. ICIE is cleared by reset.

Bit 6 OCIE Output Compare Interrupt Enable
If set, a timer interrupt is enabled whenever the OCF status flag is set; if clear, the interrupt is inhibited. OCIE is cleared by reset.

Bit 5 TOIE Timer Overflow Interrupt Enable
If set, a timer interrupt is enabled whenever the timer status flag, TOF, is set; if clear, the interrupt is inhibited. TOIE is cleared by reset.

Bit 4 EOE External Oscillator Enable (CDP68HC05D2 only)
If set, the external timer oscillator is enabled. If cleared, the inverter between pins 29 and 30 is disabled and cannot be used in a crystal or RC oscillator. EOE is cleared by reset.

Bit 3 ECC External Clock Connect (CDP68HC05D2 only)
If set, the internal clock input to the timer is disabled and the external timer oscillator is connected to the input to the timer. The ECC bit is cleared by reset.

Bit 1 IEDG Input Edge
Controls which level transition on pin 37 will trigger a free-running counter transfer to the input capture register. Reset does not affect the IEDG bit.
0 = negative edge; 1 = positive edge.

Bit 0 OLVL Output Level
This is the next value to be clocked to the output level register by a successful output compare and appears at pin 35. This bit and the output level register are cleared by reset.
0 = low output; 1 = high output.

Timer Status Register (TSR)

A timer status register (TSR) is an 8-bit register in which the three most significant bits contain read-only status information. These three bits indicate the following:

1. A proper transition has taken place at pin 37 with an accompanying transfer of the free-running counter contents to the input capture register,
2. A match has been found between the free-running counter and the output compare register, and
3. A free-running counter transition from $FFFF to $0000 has been sensed (timer overflow).

The timer status register is illustrated below followed by a definition of each bit.

7	6	5	4	3	2	1	0
ICF	OCF	TOF	0	0	0	0	0

Bit 7 ICF The input capture flag (ICF) is set when a proper edge has been sensed by the input capture edge detector. It is cleared by a processor access of the timer status register (with ICF set) followed by accessing the low byte of the input capture register. Reset does not affect the input compare flag.

Bit 6 OCF The output compare flag (OCF) is set when the output compare register contents match the contents of the free-running counter. The OCF is cleared by accessing the timer status register (with OCF set) and then accessing the low byte of the output compare register. Reset does not affect the output compare flag.

Bit 5 TOF The timer overflow flag (TOF) bit is set by a transition of the free-running counter from $FFFF to $0000. It is cleared by accessing the timer status register (with TOF set) followed by an access of the free-running counter's least significant byte. Reset does not affect the TOF bit.

Accessing the timer status register satisfies the first condition required to clear any status bits which happen to be set during the access. The only remaining step is to provide an access of the register which is associated with the status bit. Typically, this presents no problem for the input capture and output compare functions.

A problem can occur, however, when using the time overflow function and reading the free-running counter at random times to measure an elapsed time. Without incorporating the proper precautions into software, the timer overflow flag could unintentionally be cleared if: 1) the timer status register is read or written when TOF is set, and 2) the least significant byte of the free-running counter is read but not for the purpose of servicing the flag. The counter alternate register contains the same value as the free-running counter; therefore, this alternate register can be read at any time without affecting the timer overflow flag in the timer status register.

During STOP and WAIT instructions, the programmable timer functions as follows: during the wait mode, the timer continues to operate normally

and may generate an interrupt to trigger the CPU out of the wait state; during the stop mode, the timer holds at its current state, retaining all data, and resumes operation from this point when an external interrupt is received, unless the CDP68HC05D2 is set up to use the timer oscillator, in which case, the timer continues to count.

The on-chip oscillator circuit is functionally equivalent to a Schmitt NAND gate with a tri-statable output. Reset forces the oscillator into its high impedance state with a weak pull-up device on both its input and output. When the oscillator is enabled for the first time, its output is driven and the weak pull-up devices are turned off. If the oscillator is subsequently disabled, its output will be pulled high but cannot be put in a high impedance state again unless the microcomputer is reset. However, when enabled, the oscillator has no hysteresis. The feedback circuit which causes hysteresis is only switched in when the clock input to the timer is connected to the external oscillator.

The EOE (External Oscillator Enable) and ECC (External Clock Connect) bits in the Timer Control Register control the external timer oscillator.

1. Crystal Oscillator Operation —
   a. First set the EOE bit to start the crystal oscillating.
   b. When oscillation has stabilized, the ECC bit can be set to begin clocking the timer with the external timer oscillator.
2. RC Oscillator Operation —
   When it is desired to clock the timer from the timer oscillator, set both the EOE and the ECC bits at the same time. (If EOE is set before ECC, the oscillator will be biased at its midpoint in a high current state causing power to be dissipated.)
3. No external timer oscillator being used —
   If the EOE bit is never set, the oscillator will remain in its high impedance state allowing its pins to be used as PD0 and PD1 input lines. In this case, these pins function as normal inputs and should not be left floating.
4. Timer oscillator used for event counting —
   Set both the EOE and ECC bits and drive the timer oscillator input pin with the event signal which is to be counted. (Note that if EOE remains reset and only ECC is set, the event signal can be connected to the timer oscillator output pin, and the input pin can be used as a Port D input line.)

External Oscillator Input on the CDP68HC05D2

The CDP68HC05D2 features an on-chip oscillator for the timer. The timer may run using the internal oscillator or the timer oscillator. External clock connect, bit 3, in the timer control register controls the source of the timer clock. If the ECC bit is set the internal clock input to the timer is disabled and the timer oscillator is connected to the input to the timer. Because this mode of operation permits the timer to continue running when the processor is in the STOP mode, timer interrupts, if enabled, will still occur and can be used to exit from the STOP mode. Figure 35 shows the timer oscillator controls. The frequency of the external oscillator must be less than half the CPU oscillator frequency.

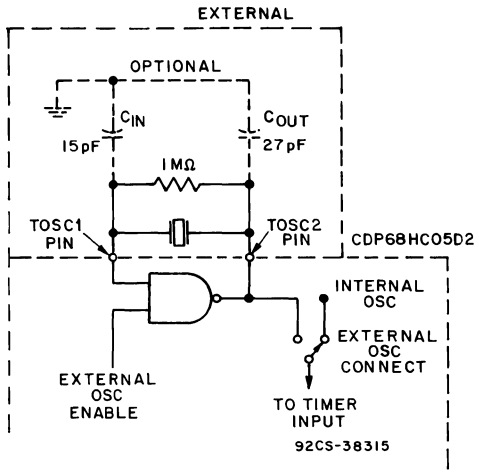


Figure 35 - External Timer Oscillator Controls

Serial Communications Interface (SCI) Featured on the CDP68HC05C4

Introduction

A full-duplex asynchronous serial communications interface (SCI) is provided on the CDP68HC05C4 with a standard NRZ format and a variety of baud rates. The SCI transmitter and receiver are functionally independent, but use the same data format and bit rate. The serial data format is standard mark/space (NRZ) which provides one start bit, eight or nine data bits, and one stop bit. "Baud" and "bit rate" are used synonymously in the following description.

SCI Two-Wire System Features

- Standard NRZ (mark/space) format.
- Advanced error detection method includes noise detection for noise duration of up to 1/16 bit time.
- Full-duplex operation (simultaneous transmit and receive).
- Software programmable for one of 32 different baud rates.
- Software transmitter and receiver enable bits.
- Separate selectable word length (eight or nine bit words).
- SCI may be interrupt driven.
- Four separate enable bits available for interrupt control.

SCI Receiver Features

- Receiver wake-up function (idle or address bit).
- Idle line detect.
- Framing error detect.
- Noise detect.
- Overrun detect.
- Receiver data register full flag.

SCI Transmitter Features

- Transmit data register empty flag.
- Transmit complete flag.
- Break send.

Any SCI two-wire system requires receive data in (RDI) and transmit data out (TDO).

Data Format

Receive data in (RDI) or transmit data out (TDO) is the serial data which is presented between the internal data bus and the output pin (TDO), and between the input pin (RDI) and the internal data bus. Data format is as shown for the NRZ in Figure 36 and must meet the following criteria:

1. A high level indicates a logic one and a low level indicates a logic zero.
2. The idle line is in a high (logic one) state prior to transmission/reception of a message.
3. A start bit (logic zero) is transmitted/received indicating the start of a message.
4. The data is transmitted and received least-significant-bit first.
5. A stop bit (high in the tenth or eleventh bit position) indicates the byte is complete.
6. A break is defined as the transmission or reception of a low (logic zero) for some multiple of the data format.

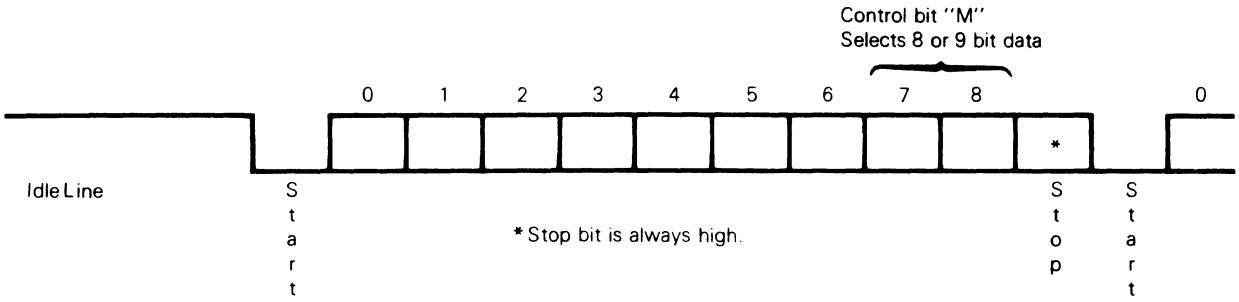


Figure 36 - Data Format

Wake-Up Feature

In a typical multiprocessor configuration, the software protocol will usually identify the addressee(s) at the beginning of the message. In order to permit uninterested MPUs to ignore the remainder of the message, a wake-up feature is included whereby all further SCI receiver flag (and interrupt) processing can be inhibited until its data line returns to the idle state. An SCI receiver is re-enabled by an idle string of at least ten (or eleven) consecutive ones. Software for the transmitter must provide for the required idle string between consecutive messages and prevent it from occurring within messages.

The user is allowed a second method of providing the wake-up feature in lieu of the idle string discussed above. This method allows the user to insert a logic one in the most significant bit of the transmit data word which needs to be received by all "sleeping" processors.

Receive Data In

Receive data in is the serial data which is presented from the input pin via the serial communications interface (SCI) to the internal data bus. While waiting for a start bit, the receiver samples the input at a rate which is 16 times higher than the set baud rate. This 16 times higher-than-baud rate is referred to as the RT rate in Figures 37 and 38, and as the receiver clock in Figure 42. When the input (idle) line is detected low, it is tested for three more sample times (referred to as the start edge verification samples in Figure 37). If at least two of these three verification samples detect a logic low, a valid start bit is assumed to have been detected (by a logic low following the three start qualifiers) as shown in Figure 37; however, if in two or more of the verification samples a logic high is detected, the line is assumed to be idle. A noise flag is set if one of the three verification samples detects a logic high; thus a valid start bit could be assumed and a noise flag still set. The receiver clock generator is controlled by the baud rate register (see Figures 41 and 42); however, the serial communications interface is synchronized by the start bit (independent of the transmitter).

Once a valid start bit is detected, the start bit, each data bit, and the stop bit are sampled three times at RT intervals of 8RT, 9RT, and 10RT (1RT is the position where the bit is expected to start) as shown

in Figure 38. The value of the bit is determined by voting logic which takes the value of the majority of samples (two or three out of three). A noise flag is set when all three samples on a valid start bit or a data bit or the stop bit do not agree. As discussed above, a noise flag is also set when the start bit verifications samples do not agree.

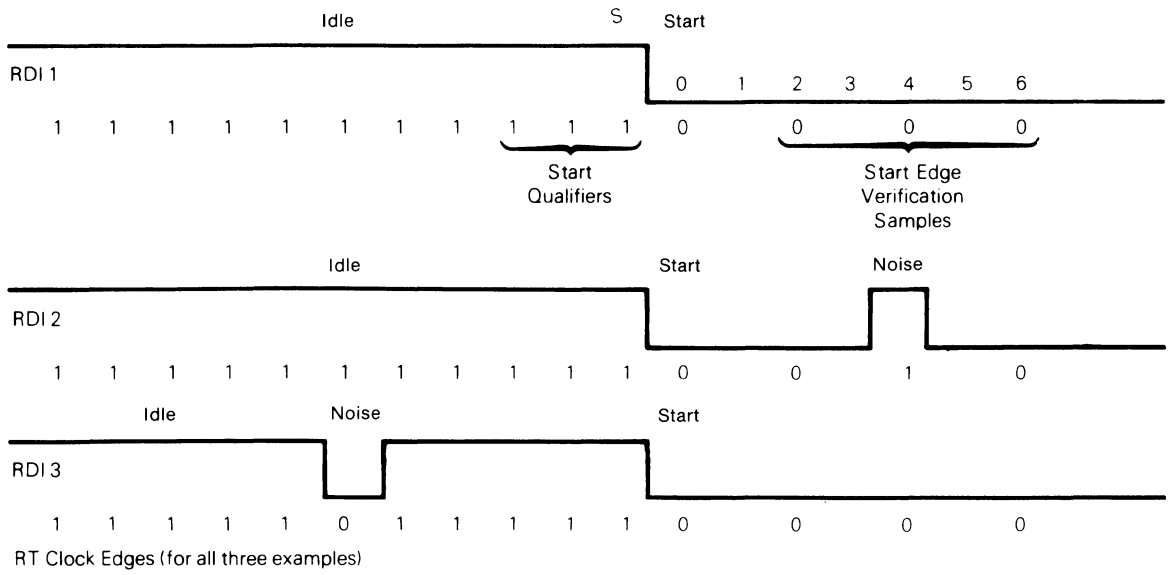


Figure 37 - Examples of Start Bit Sampling Technique

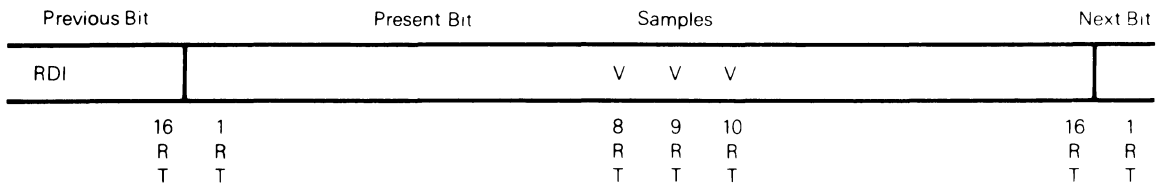
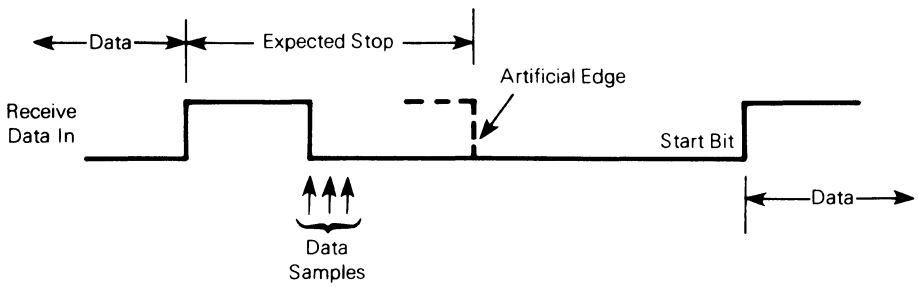


Figure 38 - Sampling Techniques Used on All Bits

Start Bit Detection Following a Framing Error

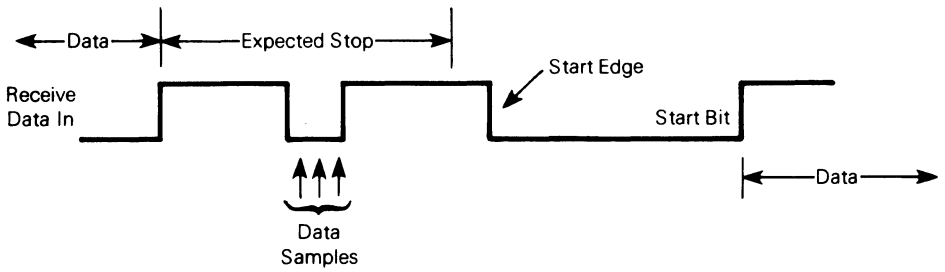
If there has been a framing error without detection of a break (10 zeros for 8-bit format or 11 zeros for 9-bit format), the circuit continues to operate as if there actually were a stop bit and the start edge will be placed artificially. The last bit received in the data shift register is inverted to a logic one, and the three logic one start qualifiers (shown in Figure 37) are forced into the sample shift register during the interval when detection of a start bit is anticipated (see Figure 39); therefore the start bit will be accepted no sooner than it is anticipated.

If the receiver detects that break (RDRF=1, FE=1, receiver data register=$00) produced the framing error, the start bit will not be artificially induced and the receiver must actually receive a logic one bit before start. See Figure 40.



(a) Case 1, Receive Line Low During Artificial Edge

Figure 39 - SCI Artificial Start Following a Framing Error



(b) Case 2, Receive Line High During Expected Start Edge

Figure 39 - SCI Artificial Start Following a Framing Error

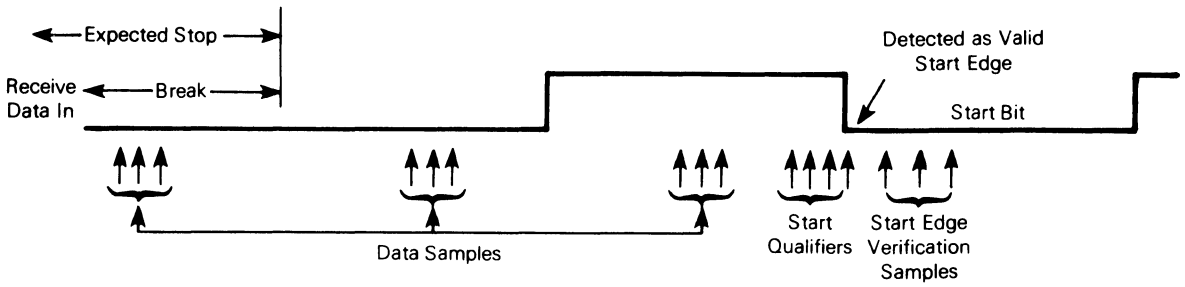


Figure 40 - SCI Start Bit Following a Break

Transmit Data Out (TDO)

Transmit data out is the serial data which is presented from the internal data bus via the serial communications interface (SCI) to the output pin. Data format is as discussed above and shown in Figure 36. The transmitter generates a bit time by using a derivative of the RT clock, thus producing a transmission rate equal to 1/16 that of the receiver sample clock.

Registers

There are five different registers used in the serial communications interface (SCI) and the internal configuration of these registers is discussed in the following paragraphs. A block diagram of the SCI system is shown in Figure 41.

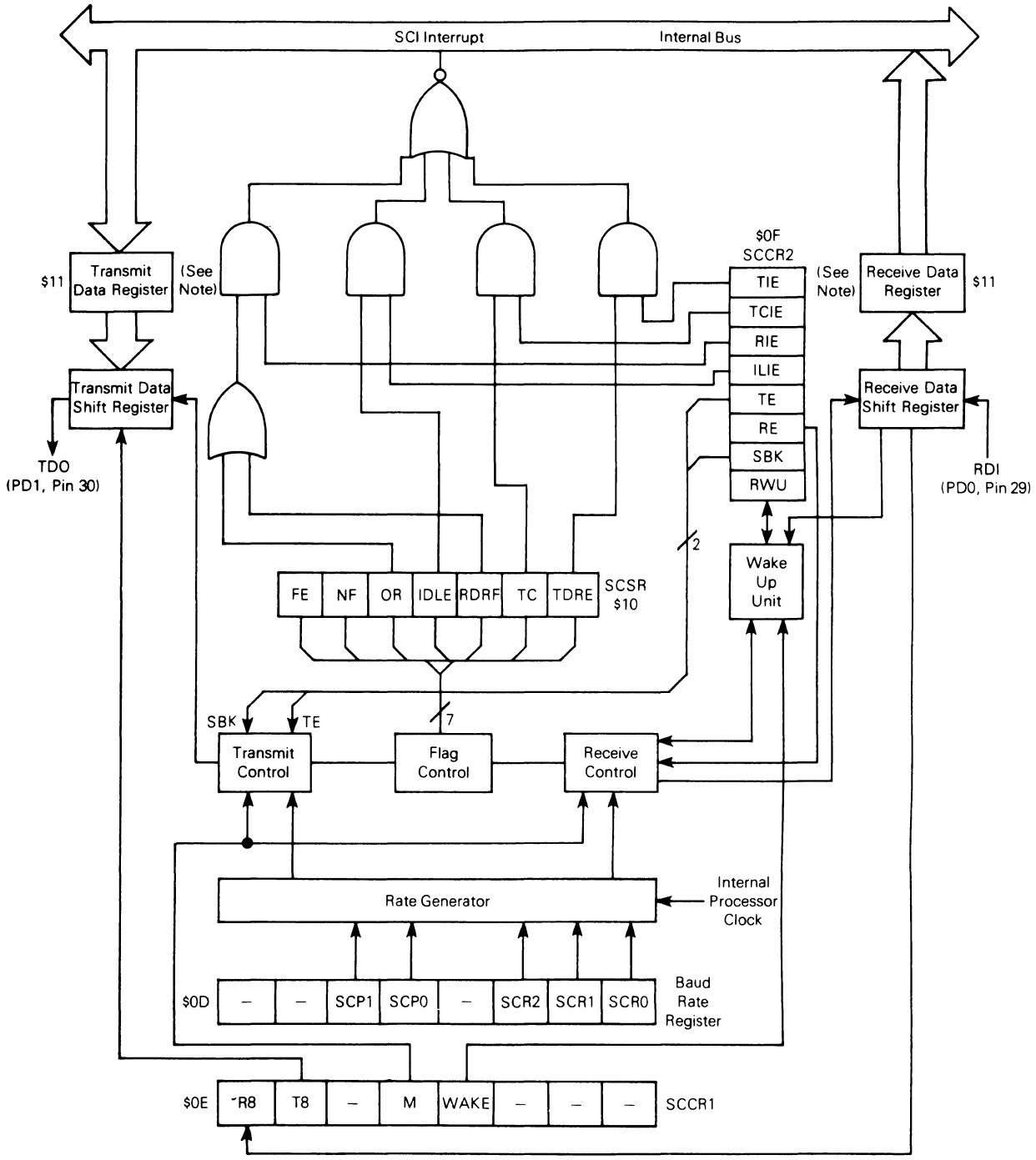
Serial Communications Data Register (SCDAT)

7	6	5	4	3	2	1	0
Serial Communications Data Register							

The serial communications data register (SCDAT) performs two functions in the serial communications interface; i.e., it acts as the receive data register when it is read and as the transmit data register when it is written. Figure 41 shows this register as two separate registers, namely: the receive data register (RDR) and the transmit data register (TDR). As shown in Figure 41 the TDR (transmit data register) provides the parallel interface from the internal data bus to the transmit shift register, and the receive data register (RDR) provides the interface from the receive shift register to the internal data bus.

When SCDAT is read, it becomes the receive data register and contains the last byte of data received. The receive data register, represented above, is a read-only register containing the last byte of data received from the shift register for the internal data bus. The RDRF bit (receive data register full bit in the serial communications status register) is set to indicate that a byte has been transferred from the input serial shift register to the serial communications data register. The transfer is synchronized with the receiver bit rate clock (from the receive control) as shown in Figure 41. All data is received, least-significant-bit first.

When SCDAT is written, it becomes the transmit data register and contains the next byte of data to be transmitted. The transmit data register, also represented above, is a write-only register containing the next byte of data to be applied to the transmit shift register from the internal data bus. As long as the transmitter is enabled, data stored in the serial communications data register is transferred to the transmit shift register (after the current byte in the shift register has been transmitted). The transfer from the SCDAT to the transmit shift register is synchronized with the bit rate clock (from the transmit control) as shown in Figure 41. All data is transmitted least-significant-bit first.



NOTE: The Serial Communications Data Register (SCDAT) is controlled by the internal R/W signal. It is the transmit data register when written and receive data register when read.

Figure 41 - Serial Communications Interface Block Diagram

Serial Communications Control Register 1 (SCCR1)

7	6	5	4	3	2	1	0
R8	T8	—	M	WAKE	—	—	—

The serial communications control register 1 (SCCR1) provides the control bits which: 1) determine the word length (either 8 or 9 bits), and 2) select the method used for the wake-up feature. Bits 6 and 7 provide a location for storing the ninth bit for longer bytes.

Bit 7 R8 — If the M bit is a one, then this bit provides a storage location for the ninth bit in the receive data byte. Reset does not affect this bit.

Bit 6 T8 — If the M bit is a one, then this bit provides a storage location for the ninth bit in the transmit data byte. Reset does not affect this bit.

Bit 4 M — The option of the word length is selected by the configuration of this bit and is shown below. Reset does not affect this bit.

0 = 1 start bit, 8 data bits, 1 stop bit
1 = 1 start bit, 9 data bits, 1 stop bit

Bit 3 WAKE — This bit allows the user to select the method for receiver "wake up". If the WAKE bit is a logic zero, an idle line condition will "wake up" the receiver. If the WAKE bit is set to a logic one, the system acknowledges an address bit (most significant bit). The address bit is dependent on both the WAKE bit and the M bit level (table shown below).

Additionally, the receiver does not use the wake-up feature unless the RWU control bit in serial communications control register 2 is set as discussed below. Reset does not affect this bit.

Wake	M	Method of Receiver "Wake-Up"
0	X	Detection of an idle line allows the next data byte received to cause the receive data register to fill and produce an RDRF flag.
1	0	Detection of a received one in the eighth data bit allows an RDRF flag and associated error flags.
1	1	Detection of a received one in the ninth data bit allows an RDRF flag and associated error flags.

Serial Communications Control Register 2 (SCCR2)

7	6	5	4	3	2	1	0
TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

The serial communications control register 2 (SCCR2) provides the control bits which individually enable/disable the transmitter or receiver, enable the system interrupts, and provide the wake-up enable bit and a "send break code" bit. Each of these bits is described below. The individual flags are discussed in the description of the serial communications status register.

Bit 7 TIE — When the transmit interrupt enable (TIE) bit is set, the SCI interrupt occurs provided the transmit data register empty (TDRE) bit is set (Figure 41). When TIE is clear, the TDRE interrupt is disabled. Reset clears the TIE bit.

Bit 6 TCIE — When the transmission complete interrupt enable (TCIE) bit is set, the SCI interrupt occurs provided the transmit complete (TC) bit is set (see Figure 41). When TCIE is clear, the TC interrupt is disabled. Reset clears the TCIE bit.

Bit 5 RIE — When the receive interrupt enable (RIE) is set, the SCI interrupt occurs provided the overrun (OR) error bit or receive data register full (RDRF) bit is set (see Figure 41). When RIE is clear, the OR and RDRF interrupts are disabled. Reset clears the RIE bit.

Bit 4 ILIE — When the idle line interrupt enable (ILIE) bit is set, the SCI interrupt occurs provided IDLE is set (see Figure 41). When ILIE is clear, the IDLE interrupt is disabled. Reset clears the ILIE bit.

Bit 3 TE — When the transmit enable (TE) bit is set, the transmit shift register output is applied to the TDO line. Depending on the state of control bit M in serial communications control register 1, a preamble of 10 (M=0) or 11 (M=1) consecutive ones is transmitted when software sets the TE bit from a cleared state. If a transmission is in progress, and TE is written to a zero, then the transmitter will wait until after the present byte had been transmitted before placing the TDO pin in the idle high-impedance state. If the TE bit has been written to a zero and then set to one before the

current byte is transmitted, the transmitter will wait until that byte is transmitted and will then initiate transmission of a new preamble. After the preamble is transmitted, and provided the TDRE bit is set (no new data to transmit), the line remains idle (driven high while TE=1); otherwise, normal transmission occurs. This function allows the user to "neatly" terminate a transmission sequence. After loading the last byte in the serial communications data register and receiving the interrupt from transmit data register empty (TDRE), indicating the data has been transferred into the shift register, the user should clear TE. The last byte will then be transmitted and the line will go idle (high impedance). Reset clears the TE bit.

Bit 2 RE When the receive enable (RE) bit is set, the receiver is enabled. When RE is clear, the receiver is disabled and all of the status bits associated with the receiver (RDRF, IDLE, OR, NF, and FE) are inhibited. Reset clears the RE bit.

Bit 1 RWU When the receiver wake-up (RWU) bit is set, it enables the "wake up" function. The type of "wake up" mode for the receiver is determined by the WAKE bit discussed above in the SCCR1. When the RWU bit is set, no status flags will be set. Flags which were set previously will not be cleared when RWU is set. If the WAKE bit is cleared, RWU is cleared after receiving 10 (M=0) or 11 (M=1) consecutive ones. Under these conditions, RWU cannot be set if the line is idle. If the WAKE bit is set, RWU is cleared after receiving an address bit. The receive data register full (RDRF) flag will then be set and the address byte will be stored in the receiver data register. Reset clears the RWU bit.

Bit 0 SBK When the send break (SBK) is set, the transmitter sends zeros in some number equal to a multiple of the data format bits. If the SKB bit is toggled set and clear, the transmitter sends 10 (M=0) or 1 (M=1) zeros and then reverts to idle or sending data. The actual number of zeros sent when SBK is toggled depends on the data format set by the M bit in the serial communications control register 1; therefore, the break code will be synchronous with respect to the data stream. At the completion of the break code, the transmitter sends at least one high bit to guarantee recognition of a valid start bit. Reset clears the SBK bit.

Serial Communications Status Register (SCSR)

7	6	5	4	3	2	1	0
TDRE	TC	RDRF	IDLE	OR	NF	FE	—

The serial communications status register (SCSR) provides inputs to the interrupt logic circuits for generation of the SCI system interrupt. In addition, a noise flag bit and a framing error bit are also contained in the SCSR.

Bit 7 TDRE The transmit data register empty (TDRE) bit is set to indicate that the contents of the serial communications data register have been transferred to the transmit serial shift register. If the TDRE bit is clear, it indicates that the transfer has not yet occurred and a write to the serial communications data register will overwrite the previous value. The TDRE bit is cleared by accessing the serial communications status register (with TDRE set), followed by writing to the serial communications data register. Data can not be transmitted unless the serial communications status register is accessed before writing to the serial communications data register to clear the TDRE flag bit. Reset sets the TDRE bit.

Bit 6 TC The transmit complete (TC) bit is set at the end of a data frame, preamble, or break condition if:

1. Transmit enable (TE) = 1, transmit data register empty (TDRE) = 1, and no pending data, preamble, or break is to be transmitted; or
2. Transmit enable (TE) = 0, and the data, preamble, or break (in the transmit shift register) has been transmitted.

The TC bit is a status flag which indicates that one of the above conditions has occurred. The TC bit is cleared by accessing the serial communications status register (with

TC set), followed by writing to the serial communications data register. It does not inhibit the transmitter function in any way. Reset sets the TC bit.

Bit 5 RDRF When the receive data register full (RDRF) bit is set, it indicates that the receiver serial shift register is transferred to the serial communications data register. If multiple errors are detected in any one received word, the noise flag (NF), framing error (FE), and RDRF bits will be affected as appropriate during the same clock cycle. The RDRF bit is cleared when the serial communications status register is accessed (with RDRF set) followed by a read of the serial communications data register. Reset clears the RDRF bit.

Bit 4 IDLE When the idle line detect bit is set, it indicates that a receiver idle line is detected (receipt of a minimum number of ones to constitute the number of bits in the byte format). The minimum number of ones needed will be 10 (M=0) or 11 (M=1). This allows a receiver that is not in the wake-up mode to detect the end of a message, detect the preamble of a new message, or to resynchronize with the transmitter. The IDLE bit is cleared by accessing the serial communications status register (with IDLE set) followed by a read of the serial communications data register. The IDLE bit will not be set again until after the receive data register full (RDRF) bit has been set; i.e., a new idle line occurs. The IDLE bit is not set by an idle line when the receiver "wakes up" from the wake-up mode. Reset clears the IDLE bit.

Bit 3 OR When the overrun (OR) error bit is set, it indicates that the next byte is ready to be transferred from the receive shift register to the serial communications data register when it is already full (RDRF bit is set). Data transfer is then inhibited until the RDRF bit is cleared. Data in the serial communications data register is valid in this case, but additional data received during an overrun condition (including the byte causing the overrun) will be lost. The OR bit is cleared when the serial communications status register is accessed (with OR set), followed by a read of the serial communications data register. Reset clears the OR bit.

Bit 2 NF The noise flag (NF) bit is set if there is noise on a "valid" start bit or if there is noise on any of the data bits or if there is noise on the stop bit. It is not set by noise on the idle line nor by invalid (false) start bits. If there is noise, the NF bit is not set until the receive data register full (RDRF) flag is set. Each data bit is sampled three times as described above in RECEIVE DATA IN and shown in Figure 38. The NF bit represents the status of the byte in the serial communications data register. For the byte being received (shifted in) there will also be a "working" noise flag, the value of which will be transferred to the NF bit when the serial data is loaded into the serial communications data register. The NF bit does not generate an interrupt because the RDRF bit gets set with NF and can be used to generate the interrupt. The NF bit is cleared when the serial communications status register is accessed (with NF set), followed by a read of the serial communications data register. Reset clears the NF bit.

Bit 1 FE The framing error (FE) bit is set when the byte boundaries in the bit stream are not synchronized with the receiver bit counter (generated by a "lost" stop bit). The byte is transferred to the serial communications data register and the receive data register full (RDRF) bit is set. The FE bit does not generate an interrupt because the RDRF bit is set at the same time as FE and can be used to generate the interrupt. Notice that if the byte received causes a framing error and it will also cause an overrun if transferred to the serial communications data register, then the overrun bit will be set, but not the framing error bit, and the byte will not be transferred to the serial communications data register. The FE bit is cleared when the serial communications status register is accessed (with FE set)

followed by a read of the serial communications data register. Reset clears the FE bit.

Baud Rate Register

7	6	5	4	3	2	1	0
—	—	SCP1	SCP0	—	SCR2	SCR1	SCR0

The baud rate register provides the means for selecting different baud rates which may be used as the rate control for the transmitter and receiver. The SCP0-SCP1 bits function as prescaler for the SCR0-SCR2 bits. Together these five bits provide multiple baud rate combinations for a given crystal frequency.

Bit 5 SCP1
Bit 4 SCP0

These two bits in the baud rate register are used as a prescaler to increase the range of standard baud rates controlled by the SCR0-SCR2 bits. A table of the prescaler internal processor clock division versus bit levels is provided below. Reset clears SCP1-SCP0 bits (divide-by-one).

SCP1	SCP0	Internal Processor Clock Divide By
0	0	1
0	1	3
1	0	4
1	1	13

Bit 2 SCR2
Bit 1 SCR1
Bit 0 SCR0

These three bits in the baud rate register are used to select the baud rates of both the transmitter and receiver. A table of baud rates versus bit levels is shown below. Reset does not affect the SCR2-SCR0 bits.

SCR2	SCR1	SCR0	Prescaler Output Divide By
0	0	0	1
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

The diagram of Figure 42 and Tables III and IV illustrate the divided chain used to obtain the baud rate clock (transmit clock). Notice that there is a fixed rate divide-by-16 between the receive clock (RT) and the transmit clock (Tx). The actual divider chain is controlled by the combined SCP0-SCP1 and SCR0-SCR2 bits in the baud rate register as illustrated. All divided frequencies shown in the first table represent the final transmit clock (the actual baud rate) resulting from the internal processor clock division shown in the "divide-by" column only (prescaler division only). The second table illustrates how the prescaler output can be further divided by action of the SCI select bits (SCR0-SCR2). For example, assume that a 9600 Hz baud rate is required with a 2.4576 MHz external crystal. In this case the prescaler bits (SCP0-SCP1) could be configured as a divide-by-one or a divide-by-four. If a divide-by-four prescaler is used, then the SCR0-SCR2 bits must be configured as a divide-by-two. This results in a divide-by-128 of the internal processor clock to produce a 9600 Hz baud rate clock. Using the same crystal, the 9600 baud rate can be obtained with a prescaler divide-by-one and the SCR0-SCR2 bits configured for a divide-by-eight.

NOTE: The crystal frequency is internally divided-by-two to generate the internal processor clock.

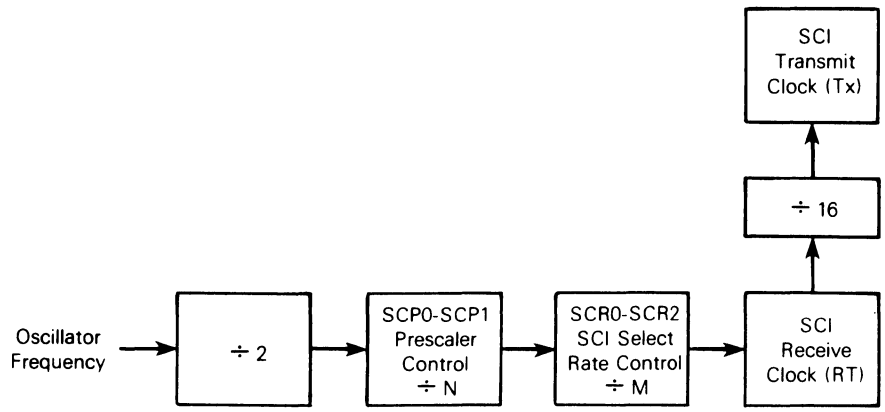


Figure 42 - Rate Generator Division

Table III - Prescaler Highest Baud Rate Frequency Output

SCP Bit		Clock*	Crystal Frequency MHz				
1	0	Divided By	4.194304	4.0	2.4576	2.0	1.8432
0	0	1	131.072 kHz	125.000 kHz	76.80 kHz	62.50 kHz	57.60 kHz
0	1	3	43.691 kHz	41.666 kHz	25.60 kHz	20.833 kHz	19.20 kHz
1	0	4	32.768 kHz	31.250 kHz	19.20 kHz	15.625 kHz	14.40 kHz
1	1	13	10.082 kHz	9600 Hz	5.907 kHz	4800 Hz	4430 Hz

*The clock in the "Clock Divided By" column is the internal processor clock.

NOTE: The divided frequencies shown in Table III represent baud rates which are the highest transmit baud rate (Tx) that can be obtained by a specific crystal frequency and only using the prescaler division. Lower baud rates may be obtained by providing a further division using the SCI rate select bits as shown below for some representative prescaler outputs.

Table IV - Transmit Baud Rate Output for a Given Prescaler Output

SCR Bits			Divide	Representative Highest Prescaler Baud Rate Output				
2	1	0	By	131.072 kHz	32.768 kHz	76.80 kHz	19.20 kHz	9600 Hz
0	0	0	1	131.072 kHz	32.768 kHz	76.80 kHz	19.20 kHz	9600 Hz
0	0	1	2	65.536 kHz	16.384 kHz	38.40 kHz	9600 Hz	4800 Hz
0	1	0	4	32.768 kHz	8.192 kHz	19.20 kHz	4800 Hz	2400 Hz
0	1	1	8	16.384 kHz	4.096 kHz	9600 Hz	2400 Hz	1200 Hz
1	0	0	16	8.192 kHz	2.048 kHz	4800 Hz	1200 Hz	600 Hz
1	0	1	32	4.096 kHz	1.024 kHz	2400 Hz	600 Hz	300 Hz
1	1	0	64	2.048 kHz	512 Hz	1200 Hz	300 Hz	150 Hz
1	1	1	128	1.024 kHz	256 Hz	600 Hz	150 Hz	75 Hz

NOTE: Table IV illustrates how the SCI select bits can be used to provide lower transmitter baud rates by further dividing the prescaler output frequency. The five examples are only representative samples. In all cases, the baud rates shown are transmit baud rates (transmit clock) and the receiver clock is 16 times higher in frequency than the actual baud rate.

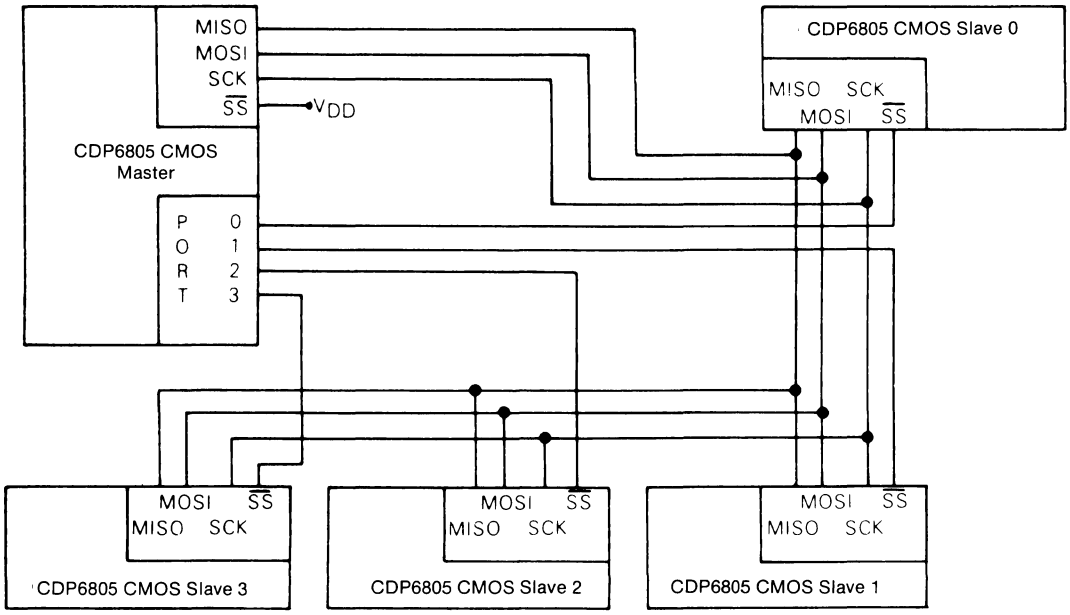
Serial Peripheral Interface (SPI)

The CDP68HC05C4 and CDP68HC05D2 microcomputers feature a serial peripheral interface (SPI) for communication with peripherals and MCUs. The SPI is a four-wire synchronous communication system with separate lines for input data, output data, clock, and slave select. A master MCU initiates the exchange of data bytes with a slave MCU, a peripheral device such as a RAM or real-time clock, or another master (which is in the slave mode during communication). In the SPI format, the clock is not included in the data stream and must be furnished as a separate signal. The master MCU produces the clocking signal to synchronize data transfer. Any device on the bus may be either a master or a slave, but at any one time, only one device is designated as the bus master. The serial bus is an efficient way to transfer data because few pins are needed.

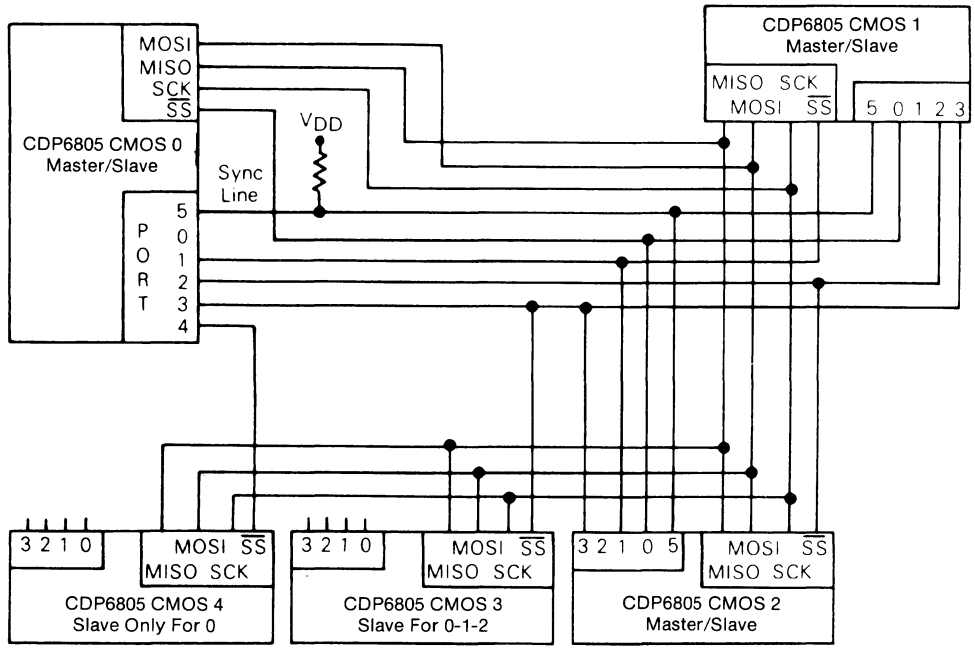
SPI Features

- Full duplex, three-wire synchronous transfers
- Master or slave operation
- 1.05 MHz (maximum) master bit frequency
- 2.1 MHz (maximum) slave bit frequency
- Four programmable master bit rates
- Programmable clock polarity and phase
- End of transmission interrupt flag
- Write collision flag protection
- Master-Master mode fault protection capability

Figure 43 illustrates two different system configurations. Figure 43a represents a system of five different MCUs in which there are one master and four slaves (0, 1, 2, 3). Figure 43b represents a system of five MCUs in which three can be master or slave and two are slave only. The SPI system transfers data synchronously over two data I/O lines, master-in/slave-out (MISO) and master-out/slave-in (MOSI), with a serial clock line (SCK) for synchronization. A slave-select ($\overline{SS}$) line is included to prevent bus contention. Notice that all $\overline{SS}$ pins are connected to a port pin of a master/slave device. In this case any of the devices can be a slave.



a. Single Master, Four Slaves



b. Three Master/Slave, Two Slaves

Figure 43 - Master-Slave System Configuration

Figure 44 shows a very simple connection diagram of a master device with a slave device. All pins of the same mnemonic are connected together on master and slave. Notice that the master $\overline{SS}$ pin is tied to a logic high and the slave $\overline{SS}$ pin is a logic low. Only the master can generate the clocking signal for data transfer to and from the slave. Each time the master loads its shift register, eight clock pulses are generated to shift this data out. As the data is shifted out of the master's shift register, it is shifted into the slave's shift register. At the same time, the data that was previously in the slave's shift register is shifted into the master's shift register. The result of this transfer is that the data in the master and slave shift registers are exchanged. In this way, a master controls data flow to and from the other devices in the system.

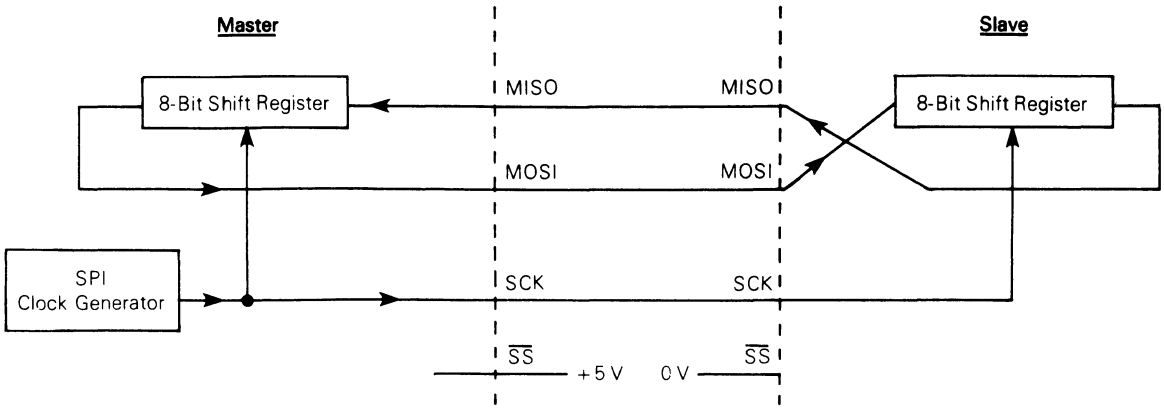


Figure 44 - Serial-Peripheral-Interface Master/Slave Interconnection

Serial Peripheral Interface Signal Description

The four basic signals (MOSI, MISO, SCK, and, $\overline{SS}$) discussed above are described in detail in the following paragraphs. Each signal function is described for both the master and slave modes.

Master Out Slave In (MOSI)

The MOSI pin is configured as a data output in a master (mode) device and as a data input in a slave (mode) device. In this manner data is transferred serially from a master to a slave on this line; most significant bit first, least significant bit last. As shown in Figure 45, four possible timing relationships may be chosen by using control bits CPOL and CPHA. The master device always allows data to be applied on the MOSI line a half-cycle before the clock edge (SCK) in order for the slave device to latch the data.

NOTE

Both the slave device(s) and a master device must be programmed to similar timing modes for proper data transfer.

When the master device transmits data to a second (slave) device via the MOSI line, the slave device

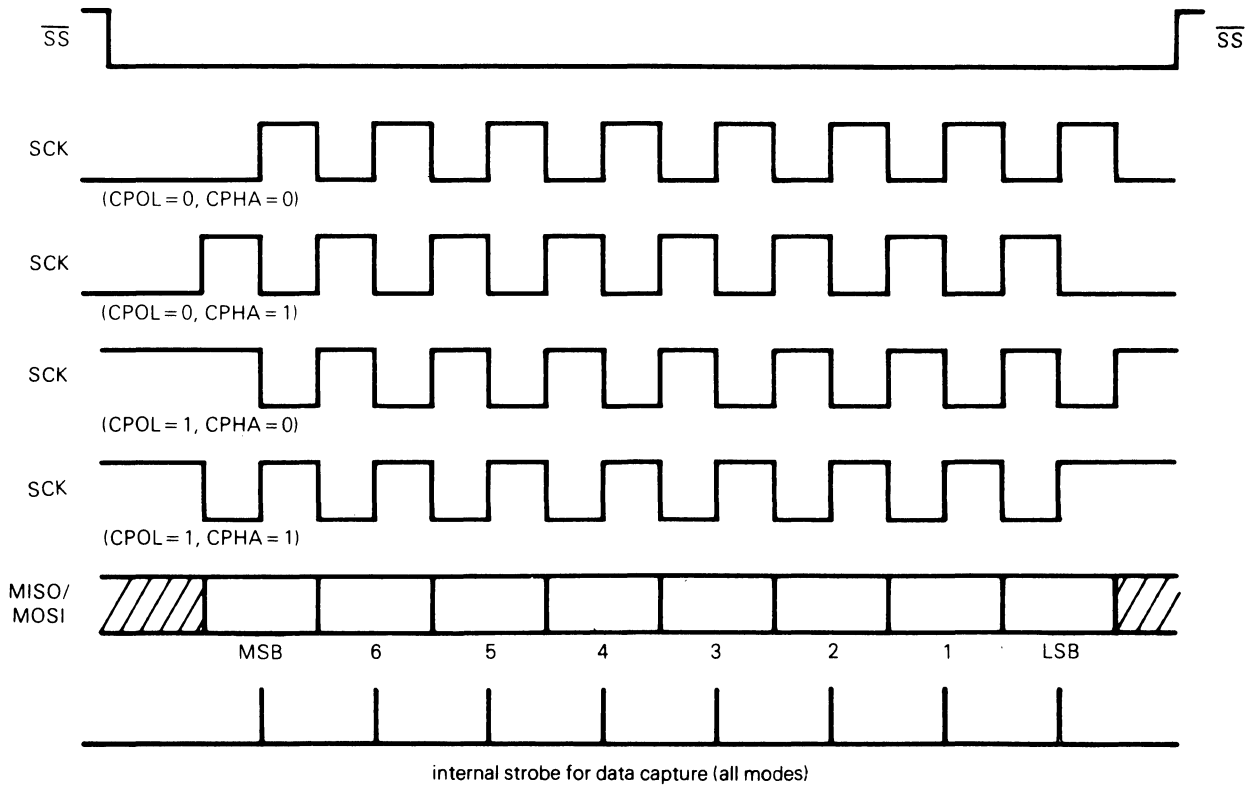


Figure 45 - Data Clock Timing Diagram

responds by sending data to the master device via the MISO line. This implies full duplex transmission with both data out and data in synchronized with the same clock signal (one which is provided by the master device). Thus, the byte transmitted is replaced by the byte received and eliminates the need for separate transmit-empty and receive-full status bits. A single status bit (SPIF) is used to signify that the I/O operation is complete.

Configuration of the MOSI pin is a function of the MSTR bit in the serial peripheral control register (SPCR). When a device is operating as a master, and the DDR bit is set in the case of the CDP68HC05D2, the MOSI pin is an output because the program in firmware sets the MSTR bit to a logic one.

Master In Slave Out (MISO)

The MISO pin is configured as an input in a master (mode) device and as an output in a slave (mode) device. In this manner, data is transferred serially from slave to a master on this line; most significant bit first, least significant bit last. The MISO pin of a slave device is placed in the high-impedance state if it is not selected by the master; i.e., its $\overline{SS}$ pin is a logic one. The timing diagram of Figure 45 shows the relationship between data and clock (SCK). As shown in Figure 45, four possible timing relationships may be chosen by using control bits CPOL and CPHA. The master device always allows data to be applied on the MOSI line a half-cycle before the clock edge (SCK) in order for the slave device to latch the data.

NOTE

The slave device(s) and a master device must be programmed to similar timing modes for proper data transfer.

When the master device transmits data to a slave device via the MOSI line, the slave device responds by sending data to the master device via the MISO line. This implies full duplex transmission with both data out and data in synchronized with the same clock signal (one which is provided by the master device). Moreover, the same shift register is used for data out and data in. Thus, the byte transmitted is replaced by the byte received and eliminates the need for separate transmit-empty and receive-full status bits. A single status bit (SPIF) in the serial peripheral status register (SPSR) is used to signify that the I/O operation is complete.

In the master device, the MSTR control bit in the serial peripheral control register (SPCR) is set to a logic one (by the program) to allow the master device to receive data on its MISO pin. In the slave device, its MISO pin is enabled by the logic level of the $\overline{SS}$ pin; i.e., if $\overline{SS}$ = 1 then the MISO pin is placed in the high-impedance state, whereas, if $\overline{SS}$ = 0 the MISO pin is an output for the slave device (only if the DDR bit is set in the case of the CDP68HC05D2).

Slave Select ($\overline{SS}$)

The slave select ($\overline{SS}$) pin is a fixed input (PD5, pin 34), which receives an active low signal that is generated by the master device to enable slave device(s) to accept data. To ensure that data will be accepted by a slave device, the $\overline{SS}$ signal line must be a logic low prior to occurrence of SCK (system clock) and must remain low until after the last (eighth) SCK cycle. Figure 45 illustrates the relationship between SCK and the data for two different level combinations of CPHA, when $\overline{SS}$ is pulled low. These are: 1) with CPHA = 1 or 0, the first bit of data (MSB) is applied to the MISO line for transfer, and 2) when CPHA = 0 the slave device is prevented from writing to its data register. Refer to the WCOL status flag in the serial peripheral status register description for further information on the effects that the $\overline{SS}$ input and CPHA control bit have on the I/O data register. The WCOL flag warns the slave if it has had a conflict between a transmission and a write of the data register. A high level $\overline{SS}$ signal forces the MISO (master in slave out) line to the high-impedance state. Also, SCK and the MOSI (master out slave in) line are ignored by a slave device when its $\overline{SS}$ signal is high.

When a device is a master, it constantly monitors its $\overline{SS}$ signal input for a logic low. The master device will become a slave device any time its $\overline{SS}$ signal input is detected low. This ensures that there is only one master controlling the $\overline{SS}$ line for a particular system. When the $\overline{SS}$ line is detected low, it clears the MSTR control bit (serial peripheral control register). Also, control bit SPE in the serial peripheral control register is cleared which causes the serial peripheral interface (SPI) to be disabled (Port D SPI pins become inputs). The MODF flag bit in the serial peripheral status register is also set to indicate to the master device that another device is attempting to become a master. Two devices attempting to be outputs are normally the result of a software error; however, a system could be configured which would contain a default master which would automatically "take over" and restart the system.

Serial Clock — SCK

The serial clock is used to synchronize the movement of data both in and out of the device through its MOSI and MISO pins. The master and slave devices are capable of exchanging a data byte of information during a sequence of eight clock pulses. Because the SCK is generated by the master device, the SCK line becomes an input on all slave devices and synchronizes slave data transfer. The type of clock and its relationship to data are controlled by the CPOL and CPHA bits in the serial peripheral control register discussed below. Refer to Figure 45 for timing.

The master device generates the SCK through a circuit driven by the internal processor clock. Two

bits (SPR0 and SPR1) in the serial peripheral control register of the master device select the clock rate. The master device uses the SCK to latch incoming slave device data on the MISO line and shifts out data to the slave device on the MOSI line. Both master and slave devices must be operated in the same timing mode as controlled by the CPOL and CPHA bit in the serial peripheral control register. In the slave device, SPR0, SPR1 have no effect on the operation of the serial peripheral interface. Timing is shown in Figure 45.

Serial Peripheral Interface Functional Description

A block diagram of the serial peripheral interface (SPI) is shown in Figure 46. A master device, once it has selected the other device(s) in the system, initiates the SCK signal by writing a byte to its SPDR. SCK is based on the internal processor clock and is used internally to control the state controller as well as the 8-bit shift register. As a master device, data is parallel-loaded into the 8-bit shift register (SPDR) from the internal bus, and shifted out serially through the MOSI pin to the slave device(s). While the shift is taking place, the master monitors the SPIF bit of the status register to determine when the data transfer is finished (SPIF=1). The data in the shift register of the slave device is simultaneously shifted out through the MISO pin, back to the master (full duplex operation). After the 8-bit shift register is loaded, its data is parallel-transferred to the read buffer and then made available to the internal data bus during a CPU read cycle. When the master requests data from the slave, the slave writes the data, which will be shifted out by the master, into its SPDR. Because the master generates the SCK signal, it must do a dummy write to its SPDR to shift the data out of the slave.

In a slave configuration, the slave start logic receives a logic low (from a master device) at the $\overline{SS}$ pin and a system clock input (from the same master device) at the SCK pin. Thus, the slave is synchronized with the master. Data from the master is received serially at the slave MOSI pin and loads the 8-bit shift register. After the 8-bit shift register is loaded, its data is parallel-transferred to the read buffer and then is made available to the internal data bus during a CPU read cycle. During a write cycle, data is parallel-loaded into the 8-bit shift register from the internal data bus and then shifted out serially to the MISO pin for application to the master device.

Serial Peripheral Interface Registers

There are three registers in the serial peripheral interface which provide control, status, and data storage functions. These registers are the serial peripheral control register (SPCR), serial peripheral status register (SPSR), and the serial peripheral data I/O register (SPDR).

Serial Peripheral Control Register (SPCR)

7	6	5*	4	3	2	1	0
SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0

*Bit 5 is a feature of the CDP68HC05D2

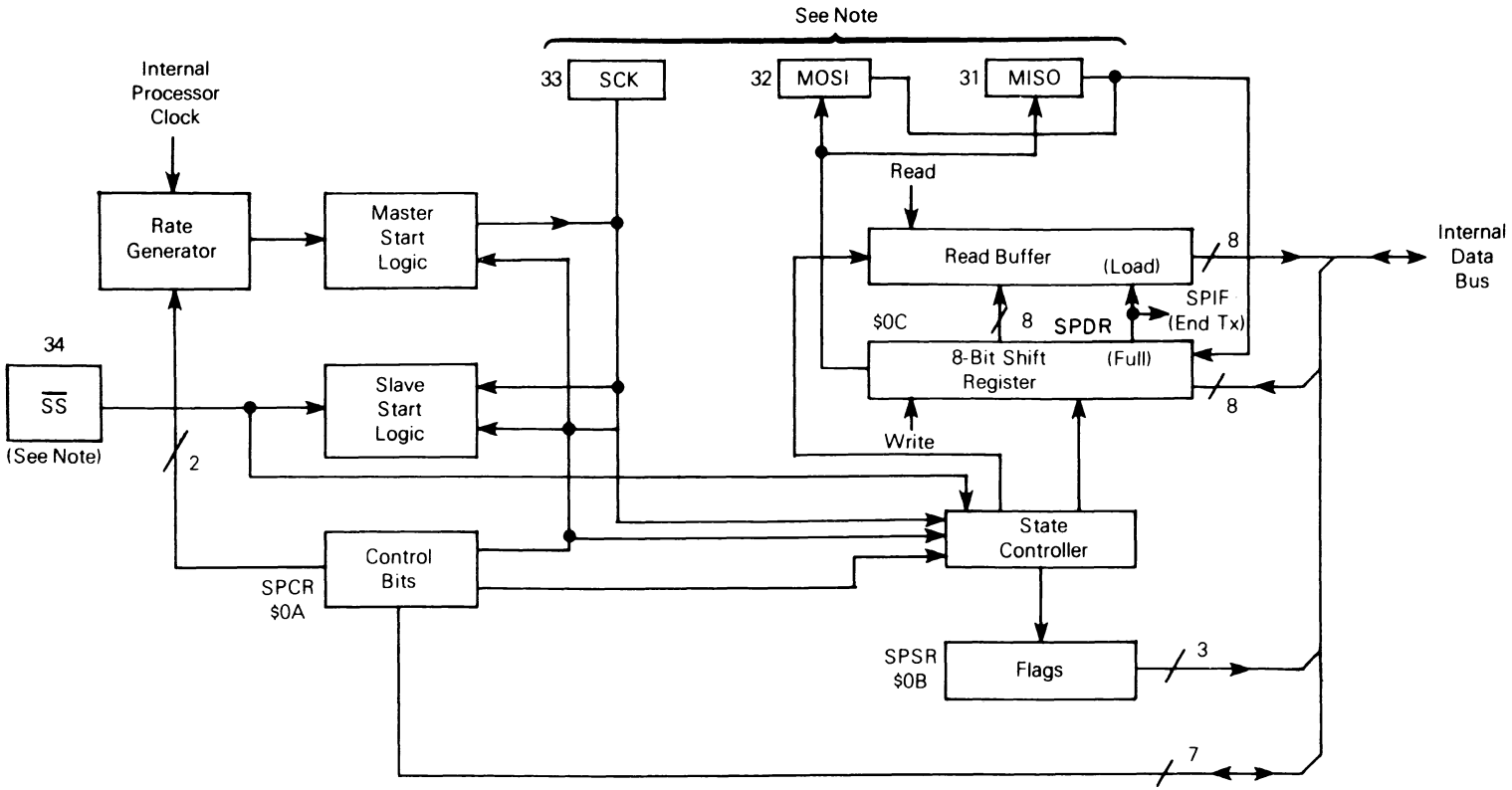
The serial peripheral control register bits are defined as follows:

Bit 7 SPIE — When the serial peripheral interrupt enable (SPIE) bit is high, it allows the occurrence of a processor interrupt, and forces the proper vector to be loaded into the program counter if the serial peripheral status register flag bit (SPIF and/or MODF) is set to a logic one. It does not inhibit the setting of a status bit. The SPIE bit is cleared by reset.

Bit 6 SPE — When the serial peripheral output enable (SPE) control bit is set, all output drive is applied to the external pins and the system is enabled. When the SPE bit is set, it enables the SPI system by connecting it to the external pins thus allowing it to interface with the external SPI bus. The pins that are defined as output depend on the mode (master or slave) of the device. Because the SPE bit is cleared by reset, the SPI system is not connected to the external pins upon reset.

Bit 5 DWOM — Port D wire-OR mode is available on the CDP68HC05D2. This bit controls the output buffers for Port D bits 2 through 5. If DWOM = 1, the four Port D output buffers behave as open-drain outputs. If DWOM = 0, the four Port D output buffers operate as normal CMOS outputs.

Bit 4 MSTR — The master bit determines whether the device is a master or a slave. A logic zero denotes a slave device and a logic one denotes a master device. If the master mode is selected, the function of the SCK pin changes from an input to an output and the functions of the MISO and MOSI pins are reversed. This allows the user to wire device pins MISO to MISO, and MOSI to MOSI, and SCK to SCK without incident. The MSTR bit is cleared by reset; therefore, the device is always placed in the slave mode during reset.



NOTE: The $\overline{SS}$, SCK, MOSI, and MISO are external pins which provide the following functions:

a. MOSI – Provides serial output to slave unit(s) when device is configured as a master. Receives serial input from master unit when device is configured as a slave unit.

b. MISO – Receives serial input from slave unit(s) when device is configured as a master. Provides serial output to master when device is configured as a slave unit.

c. SCK – Provides system clock when device is configured as a master unit. Receives system clock when device is configured as a slave unit.

d. $\overline{SS}$ – Provides a logic low to select a slave device for a transfer with a master device.

Figure 46 - Serial Peripheral Interface Block Diagram

Bit 3 CPOL The clock polarity bit controls the normal or steady state value of the clock when data is not being transferred. The CPOL bit affects both the master and slave modes. It must be used in conjunction with the clock phase control bit (CPHA) to produce the wanted clock-data relationship between a master and a slave device. When the CPOL bit is a logic zero, it produces a steady state low value at the SCK pin of the master device. If the CPOL bit is a logic one, a high value is produced at the SCK pin of the master device when data is not being transferred. The CPOL bit is not affected by reset. Refer to Figure 45.

Bit 2 CPHA The clock phase bit controls the relationship between the data on the MISO and MOSI pins and the clock produced or received at the SCK pin. This control has effect in both the master and slave modes. It must be used in conjunction with the clock polarity control bit (CPOL) to produce the wanted clock-data relation. The CPHA bit in general selects the clock edge which captures data and allows it to change states. It has its greatest impact on the first bit transmitted (MSB) in that it does or does not allow a clock transition before the first data capture edge. The CPHA bit is not affected by reset. Refer to Figure 45.

Bit 1 SPR1
Bit 0 SPR0 These two serial peripheral rate bits select one of four baud rates to be used as SCK if the device is a master; however, they have no affect in the slave mode. The slave device is capable of shifting data in and out at a maximum rate which is equal to the CPU clock. A rate table is given below for the generation of the SCK from the master. The SPR1 and SPR0 bits are not affected by reset.

SPR1	SPR0	Internal Processor Clock Divide by
0	0	2
0	1	4
1	0	16
1	1	32

Serial Peripheral Status Register (SPSR)

7	6	5	4	3	2	1	0
SPIF	WCOL	—	MODF	—	—	—	—

The status flags which generate a serial peripheral interface (SPI) interrupt may be blocked by the SPIE control bit in the serial peripheral control register. The WCOL bit does not cause an interrupt. The serial peripheral status register bits are defined as follows:

Bit 7 SPIF The serial peripheral data transfer flag bit signals the user that a data transfer between the device and an external device has been completed. With the completion of the data transfer, SPIF is set, and if SPIE is set, a serial peripheral interrupt (SPI) is generated. During the clock cycle that SPIF is being set, a copy of the received data byte in the shift register is moved to a buffer. When the data register is read, it is the buffer that is read. During an overrun condition, when the master device has sent several bytes of data and the slave device has not responded to the first SPIF, only the first byte sent is contained in the receiver buffer and all other bytes are lost.

The transfer of data is initiated by the master device writing to its serial peripheral data register.

Clearing the SPIF bit is accomplished by a software sequence of accessing the serial peripheral status register while SPIF is set, followed by a write to or a read of the serial peripheral data register. While SPIF is set, all writes to the serial peripheral data register are inhibited until the serial peripheral status register is read. This occurs in the master device. In the slave device, SPIF can be cleared (using a similar sequence) during a second transmission; however, it must be cleared before the second SPIF in order to prevent an overrun condition. The SPIF bit is cleared by reset.

Bit 6 WCOL The function of the write collision status (WCOL) bit is to signal the user that an attempt was made to write to the serial peripheral data register while a data transfer was taking place with an external device.

The transfer continues uninterrupted; therefore, a write will be unsuccessful. A "read collision" will never occur since the received data byte is placed in a buffer in which access is always synchronous with the MCU operation. If a "write collision" occurs, WCOL is set but no SPI interrupt is generated. The WCOL bit is a status flag only.

Clearing the WCOL bit is accomplished by a software sequence of accessing the serial peripheral status register while WCOL is set, followed by 1) a read of the serial peripheral data register prior to the SPIF bit being set, or 2) a read or write of the serial peripheral data register after the SPIF bit is set. A write to the serial peripheral data register (SPDR) prior to the SPIF bit being set will result in generation of another WCOL status flag. Both the SPIF and WCOL bits will be cleared in the same sequence. If a second transfer has started while trying to clear (the previously set) SPIF and WCOL bits with a clearing sequence containing a write to the serial peripheral data register, only the SPIF bit will be cleared.

A collision of a write to the serial peripheral data register while an external data transfer is taking place can occur in both the master mode and the slave mode, although with proper programming the master device should have sufficient information to preclude this collision.

Collision in the master device is defined as a write of the serial peripheral data register while the internal rate clock (SCK) is in the process of transfer. The signal on the $\overline{SS}$ pin is always high on the master device.

A collision in a slave device is defined in two separate modes. One problem arises in a slave device when the CPHA control bit is a logic zero. When CPHA is a logic zero, data is latched with the occurrence of the first clock transition. The slave device does not have any way of knowing when that transition will occur; therefore, the slave device collision occurs when it attempts to write to the serial peripheral data register after its $\overline{SS}$ pin has been pulled low. The $\overline{SS}$ pin of the slave device freezes the data in its serial peripheral data register and does not allow it to be altered if the CPHA bit is a logic zero. The master device must raise the $\overline{SS}$ pin of the slave device high between each byte it transfers to the slave device.

The second collision mode is defined for the state of the CPHA control bit being a logic one. With the CPHA bit set, the slave device will be receiving a clock (SCK) edge prior to the latch of the first data transfer. This first clock edge will freeze the data in the slave device I/O register and allow the MSB onto the external MISO pin of the slave device. The $\overline{SS}$ pin low state enables the slave device but the drive onto the MISO pin does not take place until the first data transfer clock edge. The WCOL bit will only be set if the I/O register is accessed while a transfer is taking place. By definition of the second collision mode, a master device might hold a slave device $\overline{SS}$ pin low during a transfer of several bytes of data without a problem.

A special case of WCOL occurs in the slave device. This happens when the master device starts a transfer sequence (an edge of SCK for CPHA=1; or an active $\overline{SS}$ transition for CPHA=0) at the same time the slave device CPU is writing to its serial peripheral interface data register. In this case it is assumed that the data byte written (in the slave device serial peripheral interface) is lost and the contents of the slave device read buffer become the byte that is transferred. Because the master device receives back the last byte transmitted, the master device can detect that a fatal WCOL occurred.

Because the slave device is operating asynchronously with the master device, the WCOL bit may be used as an indicator of a collision occurrence. This helps alleviate the user from a strict real-time programming effort. The WCOL bit is cleared by reset.

Bit 4 MODF The function of the mode fault flag (MODF) is defined for the master mode device. If the device is a slave device, the MODF bit will be prevented from toggling from a logic zero to a logic one; however, this does not prevent the device from being in the slave mode with the MODF bit set. The MODF bit is normally a logic zero and is set only when the master device has its $\overline{SS}$ pin pulled low. Toggling the MODF bit to a logic one affects the internal serial peripheral interface (SPI) system in the following ways:

1. MODF is set and SPI interrupt is generated if SPIE=1.
2. The SPE bit is forced to a logic zero. This blocks all output drive from the device, disables the SPI system.
3. The MSTR bit is forced to a logic zero, thus forcing the device into the slave mode.
4. Resets DDR bits for MISO, MOSI, CLR.

Clearing the MODF is accomplished by a software sequence of accessing the serial peripheral status register while MODF is set followed by a write to the serial peripheral control register. Control bits SPE and MSTR may be restored to their original set state during this clearing sequence or after the MODF bit has been cleared. Hardware does not allow the user to set the SPE and MSTR bit while MODF is a logic one unless it is during the proper clearing sequence. The MODF flag bit indicates that there might have been a multi-master conflict for system control and allows a proper exit from system operation to a reset or default system state. The MODF bit is cleared by reset.

Serial Peripheral Data I/O Register (SPDR)

7	6	5	4	3	2	1	0
Serial Peripheral Data I/O Register							

The serial peripheral data I/O register is used to transmit and receive data on the serial bus. Only a write to this register will initiate transmission/reception of another byte and this will only occur in the master device. A slave device writing to its data I/O register will not initiate a transmission. At the completion of transmitting a byte of data, the SPIF status bit is set in both the master and slave devices. A write or read of the serial peripheral data I/O register, after accessing the serial peripheral status register with SPIF set, will clear SPIF.

During the clock cycle that the SPIF bit is being set, a copy of the received data byte in the shift register is being moved to a buffer. When the user reads the serial peripheral data I/O register, the buffer is actually being read. During an overrun condition, when the master device has sent several bytes of data and the slave device has not internally responded to clear the first SPIF, only the first byte is contained in the receive buffer of the slave device; all others are lost. The user may read the buffer at any time. The first SPIF must be cleared by the time a second transfer of data from the shift register to the read buffer is initiated or an overrun condition will exist.

A write to the serial peripheral data I/O register is not buffered and places data directly into the shift register for transmission.

The ability to access the serial peripheral data I/O register is limited when a transmission is taking place. It is important to read the discussion defining the WCOL and SPIF status bits to understand the limits on using the serial peripheral data I/O register.

Serial Peripheral Interface (SPI) System Considerations

There are two types of SPI systems: single master system and multi-master systems. Figure 43 illustrates both of these systems and a discussion of each is provided below.

Figure 43a illustrates how a typical single master system may be configured, using a CDP6805 CMOS Family device as the master and four CDP6805 CMOS Family devices as slaves. As shown, the MOSI, MISO, and SCK pins are all wired to equivalent pins on each of the five devices. The master device generates the SCK clock, the slave devices all receive it. Because the CDP6805 CMOS master device is the bus master, it internally controls the function of its MOSI and MISO lines, thus writing data to the slave devices on the MOSI and reading data from the slave devices on the MISO lines. The master device selects the individual slave devices by using four pins of a parallel port to control the four $\overline{SS}$ pins of the slave devices. A slave device is selected when the master device pulls its $\overline{SS}$ pin low. The $\overline{SS}$ pins are pulled high during reset because the master device ports will be forced to be inputs at that time, thus disabling the slave devices. Notice that the slave devices do not have to be enabled in a mutually exclusive fashion except to prevent bus contention on the MISO line. For example, three slave devices enabled for a transfer are permissible if only one has the capability of being read by the master. An example of this is a write to several display drivers to clear a display with a single I/O operation. To ensure that

proper data transmission is occurring between the master device and a slave device, the master device may have the slave device respond with a previously received data byte (this data byte could be inverted or at least be a byte that is different from the last one sent by the master device). The master device will always receive the previous byte back from the slave device if all MISO and MOSI lines are connected and the slave has not written to its data I/O register. Other transmission security methods might be defined using ports for handshake lines or data bytes with command fields.

A multi-master system may also be configured by the user. A system of this type is shown in Figure 43b. An exchange of master control could be implemented using a handshake method through the I/O ports or by an exchange of code messages through the serial peripheral interface system. The major device control that plays a part in this system is the MSTR bit in the serial peripheral control register and the MODF bit in the serial peripheral status register.

CDP6805E2/E3 Microprocessor (MPU) External Bus Description

The CDP6805E2/E3 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 CDP6805E2/E3 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 CDP6805E2/E3 latches read data (R/$\overline{W}$ is high) on the falling edge of DS.

The CDP6805E2/E3 bus timing is generated from the waveform at the OSC1 input. Figure 47 shows the relationship of the CDP6805E2/E3 bus timing to the OSC1 input. Because the CDP6805E2/E3 is a completely static device, it may be operated at any frequency below its maximum (1 MHz bus) rate. Because generating the timing specifications for all of the possible frequencies is impossible, Figure 47 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}.

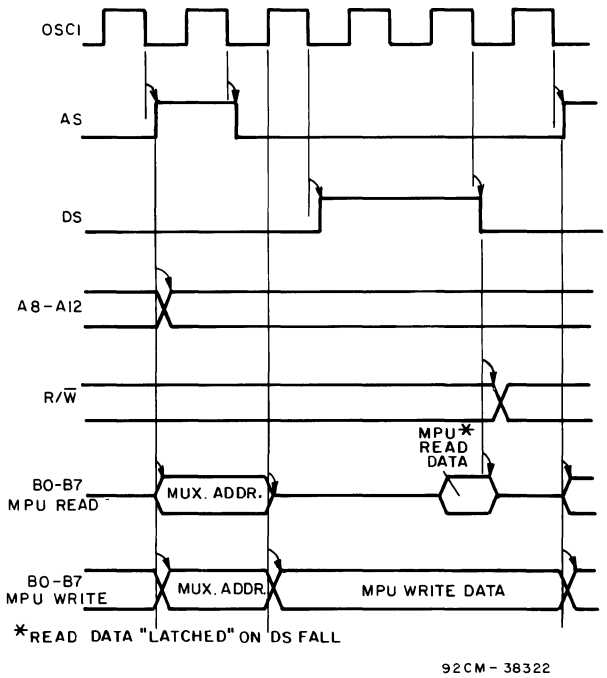


Figure 47 - OSC1 to Bus Transitions

Self-Check

Introduction

One of the advanced architectural features of the CDP6805 CMOS Family of microcomputers is the ability to test itself, using on-chip firmware. These programs are commonly referred to as self-check routines and subroutines, which are used for quick go/no-go functional tests of the individual microcomputer (MCU). The self-check routines functionally exercise the ports, RAM, ROM, timer, and interrupts, and where applicable, the SPI and SCI.

The CDP6805E2 and CDP6805E3 microprocessors do not contain on-chip ROM, and consequently do not include self-check routines.

The additional components and terminal connections necessary to support the self-check of each MCU are shown in their respective data sheets. The self-check routines are initiated by application of power to the test set-up, or by actuation of the reset switch. These self-check subroutines are initiated and automatically sequenced through their predetermined programs with individual results as noted in the data sheets.

Several of the self-check subroutines can be initiated by the user program. These user-callable subroutines are RAM, ROM, and timer (provided the timer is clocked by the internal clock) tests. One extremely valuable feature of the self-check is that it can be incorporated into the acceptance test of all CDP6805 CMOS Family devices (except CDP6805E2 and CDP6805E3) to provide go/no-go indications for the particular device.

The user-callable tests shown for the devices listed in Table V can be part of the normal power-up sequence, or included in the regular preventive maintenance schedule as well as the repair/service schedule for the user system. These self-check subroutines can be called by the user and merged into the overall system program without additional components or terminal connections. Table V contains a list of microcomputers which have this self-check firmware. Also, the address to enter each user-callable self-check subroutine is listed with appropriate comments. Each self-check subroutine ends with the RTS instruction, and must be called by the BSR/JSR instruction from the user's main program.

Table V - Subroutine Entry Addresses

MCU	RAM Test	ROM Test	Timer Test*
CDP6805F2	$078B	$07A4	$07BE
CDP6805G2	$1F87	$1FA1	$1FBB
CDP68HC05C4	—	$1F93	$1F0E
CDP68HC05D2	—	$1F93	$1F0E

(*) The timer clock source must be the internal clock.

Self-Check Description

RAM Self-Check

The RAM self-check routine performs a walking-bit diagnostic pattern, and when completed, the Z bit is cleared if any error was detected. If no error was detected, the Z bit is set. The RAM self-check routine overwrites the accumulator, index register, and RAM.

ROM Self-Check

The ROM self-check performs an exclusive OR (odd parity) checksum and returns with the Z bit clear if any error was detected. If no error was detected, the Z bit is set. Refer to the individual data sheets for RAM or registers which are modified.

Timer Self-Check

The timer self-check routine keeps track of the number of times the clock counts in some number of cycles. Because the timer has a prescaler, not every count is tested. The routine also detects a non-running timer condition. If an error is found, the Z bit is cleared; otherwise the Z bit is set indicating no error. The accumulator, the index register, and some RAM may be overwritten.

In order to work correctly as a user subroutine, the internal clock must be the clocking source, and interrupts must be disabled. At the end of the test, the clock may be running and interrupt mask cleared so the user may have to protect the program from interrupt. Refer to the individual part's data sheet for more information.

Flowchart Example

An example of the timer test for the CDP6805G2 is shown flowcharted in Figure 48. The pass/fail result can be utilized by the user program for system go/no-go considerations. Notice that previous values in the accumulator and index registers are lost.

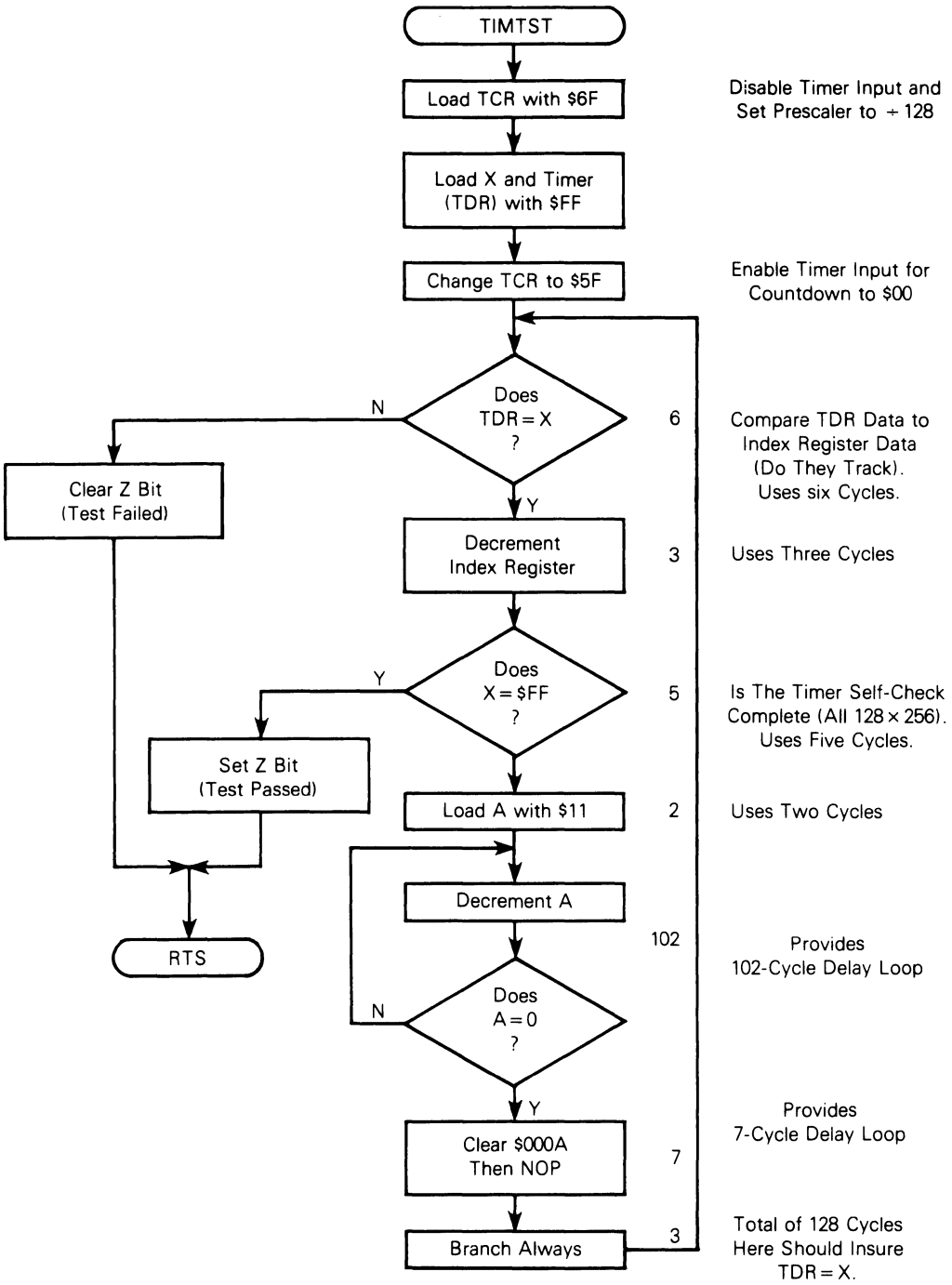


Figure 48 - CDP6805G2 MPU Timer Test (TIMTST) Flowchart

Instruction Set Detailed Definition

Introduction

In the pages that follow this section, the various accumulator and memory operations, together with the respective mnemonic, provide a heading for each of the executable instructions. 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, number of bytes required, and the opcode for each usable addressing mode. The next section contains a listing of the various nomenclature (abbreviations and signs) used in the operations.

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 (two's 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 two's 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)

Operation:
ACCA ← ACCA + M + C

Description:
Add the contents of the Carry/Borrow bit to the sum of the contents of the Accumulator and Memory, and place the result in the Accumulator.

Condition Codes:
H: Set if there is 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 is a carry from the most significant bit of the result; cleared otherwise.

Boolean Formula(e) 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	Bytes	Opcode
Immediate	2	2	A9
Direct	3	2	B9
Extended	4	3	C9
Indexed 0 Offset	3	1	F9
Indexed 1-Byte	4	2	E9
Indexed 2-Byte	5	3	D9

ADD (Add)

Operation:
ACCA ← ACCA + M

Description:
Add the contents of the Accumulator and the contents of Memory and place the result in the Accumulator.

Condition Codes:
H: Set if there is 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.

ADD (Add) (Cont'd)

C: Set if there is a carry from the most significant bit of the result; cleared otherwise.

Boolean Formula(e) 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	Bytes	Opcode
Immediate	2	2	AB
Direct	3	2	BB
Extended	4	3	CB
Indexed 0 Offset	3	1	FB
Indexed 1-Byte	4	2	EB
Indexed 2-Byte	5	3	DB

AND (Logical AND)

Operation:
ACCA ← ACCA · M

Description:
Perform logical AND between the contents of the Accumulator and the contents of Memory and place the result in the Accumulator. Each bit of the Accumulator after the operation will be the logical AND result of the corresponding bits of Memory and of the Accumulator before the operation.

Condition Codes:
H, I, C: 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.

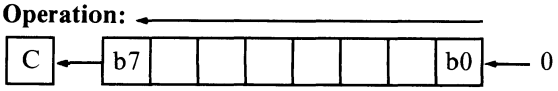
Boolean Formula(e) for Condition Codes:
$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s):
AND P

Addressing Mode	Cycles	Bytes	Opcode
Immediate	2	2	A4
Direct	3	2	B4
Extended	4	3	C4
Indexed 0 Offset	3	1	F4
Indexed 1-Byte	4	2	E4
Indexed 2-Byte	5	3	D4

ASL (Arithmetic Shift Left)

Operation:



Description:
Shift all bits of the Accumulator, Index Register, or Memory one place to the left. Bit 0 is loaded with a zero. The Carry/Borrow bit is loaded from the most significant bit of the Accumulator, Index Register, or Memory.

Condition Codes:
H, 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, were set; cleared otherwise.

Boolean Formula(e) 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)

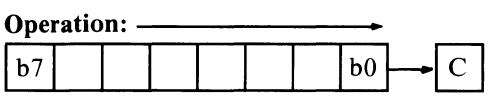
Comments:
Same opcode as LSL

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

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	48
Index Register	3	1	58
Direct	5	2	38
Indexed 0 Offset	5	1	78
Indexed 1-Byte	6	2	68

ASR (Arithmetic Shift Right)

Operation:



Description:
Shift all bits of the Accumulator, Index Register, or Memory one place to the right. Bit 7 is held constant. Bit 0 is loaded into the Carry/Borrow bit.

Condition Codes:
H, 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.

ASR (Arithmetic Shift Right) (Cont'd)

C: Set if, before the operation, the least significant bit of ACCA, X or M were set; cleared otherwise.

Boolean Formula(e) 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 = b0 (before operation)

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

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	47
Index Register	3	1	57
Direct	5	2	37
Indexed 0 Offset	5	1	77
Indexed 1-Byte	6	2	67

BCC (Branch if Carry Clear)

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

Description:
Test the state of the Carry/Borrow bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	24

BCLRn (Clear Bit in Memory)

Operation:
Mn ← 0

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

Condition Codes:
Not affected.

Source Form(s):
BCLR n, DR

Addressing Mode	Cycles	Bytes	Opcode
Direct	5	2	11 + 2n

BCS (Branch if Carry Set)

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

Description:
Test the state of the Carry/Borrow bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	25

BEQ (Branch if Equal)

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

Description:
Test the state of the Zero Indicator bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	27

BHCC (Branch if Half Carry Clear)

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

Description:
Test the state of the Half Carry Indicator bit and cause a branch if and only if H is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes:
Not affected.

BHCC (Branch if Half Carry Clear) (Cont'd)

Source Form(s):
BHCC dd

Addressing Mode	Cycles	Bytes	Opcode
Relative	3	2	28

BHCS (Branch if Half Carry Set)

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

Description:
Test the state of the Half Carry Indicator bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	29

BHI (Branch if Higher)

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

Description:
Cause a branch if and only if both Carry/Borrow and Zero Indicator 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., Accumulator) is greater than the unsigned binary number represented by the subtrahend (i.e., Memory). See BRA instruction for further details of the execution of the branch.

Condition Codes:
Not affected.

Source Form(s):
BHI dd

Addressing Mode	Cycles	Bytes	Opcode
Relative	3	2	22

BHS (Branch if Higher or Same)

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

Description:
Following an unsigned compare or subtract, BHS will cause a branch if and only if the register being compared is 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	Bytes	Opcode
Relative	3	2	24

BIH (Branch if Interrupt Line is High)

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

Description:
Test the state of the external interrupt pin and branch if and only if 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	Bytes	Opcode
Relative	3	2	2F

BIL (Branch if Interrupt Line is Low)

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

Description:
Test the state of the external interrupt pin and branch if and only if 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	Bytes	Opcode
Relative	3	2	2E

BIT (Bit Test Memory with Accumulator)

Operation:
ACCA · M

Description:
Perform the logical AND comparison of the contents of the Accumulator and the contents of Memory and modify the condition codes accordingly. The contents of the Accumulator and Memory are unchanged.

Condition Codes:
H, I, C: 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 are cleared; cleared otherwise.

Boolean Formula(e) 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	Bytes	Opcode
Immediate	2	2	A5
Direct	3	2	B5
Extended	4	3	C5
Indexed 0 Offset	3	1	F5
Indexed 1-Byte	4	2	E5
Indexed 2-Byte	5	3	D5

BLO (Branch if Lower)

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

Description:
Following a compare, BLO will branch if and only if the register being compared is 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	Bytes	Opcode
Relative	3	2	25

BMC (Branch if Interrupt Mask is Clear)

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

Description:
Test the state of the Interrupt Mask bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	2C

BLS (Branch if Lower or Same)

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

Description:
Cause a branch if Carry/Borrow is set OR Zero Indicator 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., Accumulator) is less than or equal to the unsigned binary number represented by the subtrahend (i.e., Memory). See BRA instruction for further details of the execution of the branch.

Condition Codes:
Not affected.

Source Form(s):
BLS dd

Addressing Mode	Cycles	Bytes	Opcode
Relative	3	2	23

BMI (Branch if Minus)

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

Description:
Test the state of the Negative Indicator bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	2B

BMS (Branch if Interrupt Mask Bit is Set)

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

Description:
Test the state of the Interrupt Mask bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	2D

BPL (Branch if Plus)

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

Description:
Test the state of the Negative Indicator bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	2A

BNE (Branch if Not Equal)

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

Description:
Test the state of the Zero Indicator bit and cause a branch if and only if 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	Bytes	Opcode
Relative	3	2	26

BRA (Branch Always)

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 two's 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	Bytes	Opcode
Relative	3	2	20

BRCLR n (Branch if Bit n is Clear)

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

Description:
Test bit n (n = 0-7) of memory location M and branch if and only if the bit is clear.

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

Boolean Formula(e) for Condition Codes:
C = Mn

Comments:
The Carry/Borrow 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	Bytes	Opcode
Relative	5	3	01 + 2n

BRSET n (Branch if Bit n is Set)

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

Description:
Test bit n (n = 0-7) of memory location M and branch if and only if the bit is set.

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

Boolean Formula(e) for Condition Codes:
C = Mn

Comments:
The Carry/Borrow 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):
BRSET n, DR, dd

Addressing Mode	Cycles	Bytes	Opcode
Relative	5	3	2n

BRN (Branch Never)

Description:
Never branches. Branch never is a 2 byte, 3 cycle NOP.

Condition Codes:
Not affected.

Comments:
BRN is included here to demonstrate the nature of branches on the CDP6805 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	Bytes	Opcode
Relative	3	2	21

BSET n (Set Bit in Memory)

Operation:
Mn ← 1

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

Condition Codes:
Not affected.

Source Form(s):
BSET n, DR

Addressing Mode	Cycles	Bytes	Opcode
Direct	5	2	10 + 2n

BSR (Branch to Subroutine)

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

Description:
The program counter is incremented by two. 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 ones 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	Bytes	Opcode
Relative	6	2	AD

CLC (Clear Carry Bit)

Operation:
C bit ← 0

Description:
Clear the Carry/Borrow bit in the processor condition code register.

Condition Codes:
H, I, N, Z: Not affected.
C: Cleared.

Boolean Formula(e) for Condition Codes:
C = 0

Source Form(s):
CLC

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	98

CLI (Clear Interrupt Mask Bit)

Operation:
I bit ← 0

Description:
Clear 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, N, Z, C: Not affected.
I: Cleared.

Boolean Formula(e) for Condition Codes:
I = 0

Source Form(s):
CLI

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	9A

CLR (Clear)

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

Description:
The contents of the Accumulator, Index Register, or Memory are replaced with zeroes.

Condition Codes:
H, I, C: Not affected.
N: Cleared.
Z: Set.

Boolean Formula(e) for Condition Codes:
N = 0
Z = 1

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

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	4F
Index Register	3	1	5F
Direct	5	2	3F
Indexed 0 Offset	5	1	7F
Indexed 1-Byte	6	2	6F

CMP (Compare Accumulator with Memory)

Operation:
ACCA - M

Description:
Compare the contents of the Accumulator and the contents of Memory and set the condition codes, which may then be used for controlling the conditional branches. Both operands are unaffected.

Condition Codes:
H, 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 Formula(e) 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	Bytes	Opcode
Immediate	2	2	A1
Direct	3	2	B1
Extended	4	3	C1
Indexed 0 Offset	3	1	F1
Indexed 1-Byte	4	2	E1
Indexed 2-Byte	5	3	D1

COM (Complement)

Operation:
$X \leftarrow \sim X = \$FF - X$ or,
$ACCA \leftarrow \sim ACCA = \$FF - ACCA$ or,
$M \leftarrow \sim M = \$FF - M$

Description:
Replace the contents of the Accumulator, Index Register, or Memory with the one's complement. Each bit of the operand is replaced with the complement of that bit.

Condition Codes:
H, I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.

COM (Complement) (Cont'd)

Z: Set if all bits of the result are cleared; cleared otherwise.
C: Set.

Boolean Formula(e) 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	Bytes	Opcode
Accumulator	3	1	43
Index Register	3	1	53
Direct	5	2	33
Indexed 0 Offset	5	1	73
Indexed 1-Byte	6	2	63

CPX (Compare Index Register with Memory)

Operation:
X - M

Description:
Compare the contents of Index Register to the contents of Memory and set the condition codes, which may then be used for controlling the conditional branches. Both operands are unaffected.

Condition Codes:
H, 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 Formula(e) 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	Bytes	Opcode
Immediate	2	2	A3
Direct	3	2	B3
Extended	4	3	C3
Indexed 0 Offset	3	1	F3
Indexed 1-Byte	4	2	E3
Indexed 2-Byte	5	3	D3

DEC (Decrement)

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

Description:
Subtract one from the contents of the Accumulator, Index Register, or Memory. The Negative Indicator and Zero Indicator bits are set or reset according to the result of this operation. The Carry/Borrow bit is not affected by this operation.

Condition Codes:
H, I, C: 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.

Boolean Formula(e) 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):
DEC Q, DECA, DECX (DEX is recognized by the Assembler as DECX)

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	4A
Index Register	3	1	5A
Direct	5	2	3A
Indexed 0 Offset	5	1	7A
Indexed 1-Byte	6	2	6A

EOR (Exclusive OR Memory with Accumulator)

Operation:
ACCA ← ACCA + M

Description:
Perform the logical EXCLUSIVE OR between the contents of the Accumulator and the contents of Memory, and place the result in the Accumulator. Each bit of the Accumulator after the operation will be the logical EXCLUSIVE OR of the corresponding bit of Memory and the Accumulator before the operation.

Condition Codes:
H, I, C: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.

EOR (Exclusive OR Memory with Accumulator) (Cont'd)

Z: Set if all bits of the result are cleared; cleared otherwise.

Boolean Formula(e) 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	Bytes	Opcode
Immediate	2	2	A8
Direct	3	2	B8
Extended	4	3	C8
Indexed 0 Offset	3	1	F8
Indexed 1-Byte	4	2	E8
Indexed 2-Byte	5	3	D8

INC (Increment)

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

Description:
Add one to the contents of the Accumulator, Index Register or Memory. The Negative Indicator and Zero Indicator bits are set or reset according to the result of this operation. The Carry/Borrow bit is not affected by this operation.

Condition Codes:
H, I, C: 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.

Boolean Formula(e) 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	Bytes	Opcode
Accumulator	3	1	4C
Index Register	3	1	5C
Direct	5	2	3C
Indexed 0 Offset	5	1	7C
Indexed 1-Byte	6	2	6C

JMP (Jump)

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	Bytes	Opcode
Direct	2	2	BC
Extended	3	3	CC
Indexed 0 Offset	2	1	FC
Indexed 1-Byte	3	2	EC
Indexed 2-Byte	4	3	DC

JSR (Jump to Subroutine)

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 ones 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	Bytes	Opcode
Direct	5	2	BD
Extended	6	3	CD
Indexed 0 Offset	5	1	FD
Indexed 1-Byte	6	2	ED
Indexed 2-Byte	7	3	DD

LDA (Load Accumulator from Memory)

Operation:
ACCA ← M

Description:
Load the contents of Memory into the Accumulator. The condition codes are set according to the data.

Condition Codes:
H, I, C: 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.

Boolean Formula(e) 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	Bytes	Opcode
Immediate	2	2	A6
Direct	3	2	B6
Extended	4	3	C6
Indexed 0 Offset	3	1	F6
Indexed 1-Byte	4	2	E6
Indexed 2-Byte	5	3	D6

LDX (Load Index Register from Memory)

Operation:
$X \leftarrow M$

Description:
Load the contents of Memory into the Index Register. The condition codes are set according to the data.

Condition Codes:
H, I, C: 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.

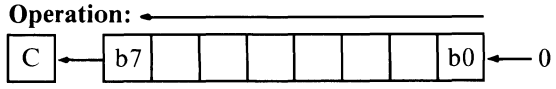
Boolean Formula(e) for Condition Codes:
$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s):
LDX P

Addressing Mode	Cycles	Bytes	Opcode
Immediate	2	2	AE
Direct	3	2	BE
Extended	4	3	CE
Indexed 0 Offset	3	1	FE
Indexed 1-Byte	4	2	EE
Indexed 2-Byte	5	3	DE

LSL (Logical Shift Left)

Operation:



Description:
Shift all bits of the Accumulator, Index Register, or Memory one place to the left. Bit 0 is loaded with a zero. The Carry/Borrow bit is loaded from the most significant bit of the Accumulator, Index Register, or Memory.

Condition Codes:
H, 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 is set; cleared otherwise.

LSL (Logical Shift Left) (Cont'd)

Boolean Formula(e) 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)

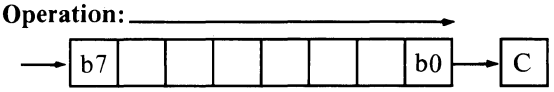
Comments:
Same as ASL

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

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	48
Index Register	3	1	58
Direct	5	2	38
Indexed 0 Offset	5	1	78
Indexed 1-Byte	6	2	68

LSR (Logical Shift Right)

Operation:



Description:
Shift all bits of the Accumulator, Index Register, or Memory one place to the right. Bit 7 is loaded with a zero. Bit 0 is loaded into the Carry/Borrow bit.

Condition Codes:
H, 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 is set; cleared otherwise.

Boolean Formula(e) for Condition Codes:
$N = R0$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$
$C = b0$ (before operation)

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

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	44
Index Register	3	1	54
Direct	5	2	34
Indexed 0 Offset	5	1	74
Indexed 1-Byte	6	2	64

MUL (Multiply)

Operation:
XA ← X*A

Description:
Multiply the eight bits in the Index Register by the eight bits in the Accumulator to obtain a 16-bit unsigned number. The most significant bits of the product are stored in the Index Register, while the least significant bits are stored in the Accumulator.

Condition Codes:
I, N, Z: Not affected.
H, C: Cleared.

Comments:
This instruction is available only on the CDP68HC05C4 and CDP68HC05D2 Microcomputers.

Source Form(s):
MUL

Addressing Mode	Cycles	Bytes	Opcode
Inherent	11	1	42

NEG (Negate)

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

Description:
Replace the contents of the Accumulator, Index Register, or Memory with its two's complement. Note that the data $80 and 00 are left unchanged by two's complement.

Condition Codes:
H, 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 are 00.

Boolean Formula(e) 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

NEG (Negate) (Cont'd)

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	40
Index Register	3	1	50
Direct	5	2	30
Indexed 0 Offset	5	1	70
Indexed 1-Byte	6	2	60

NOP (No Operation)

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	Bytes	Opcode
Inherent	2	1	9D

ORA (Inclusive OR)

Operation:
ACCA ← ACCA v M

Description:
Perform logical OR between the contents of the Accumulator and the contents of Memory and place the result in the Accumulator. Each bit of the Accumulator after the operation will be the logical (inclusive) OR result of the corresponding bits of Memory and the Accumulator before the operation.

Condition Codes:
H, I, C: 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.

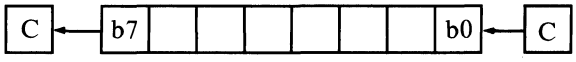
Boolean Formula(e) for Condition Codes:
N = R7
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s):
ORA P

Addressing Mode	Cycles	Bytes	Opcode
Immediate	2	2	AA
Direct	3	2	BA
Extended	4	3	CA
Indexed 0 Offset	3	1	FA
Indexed 1-Byte	4	2	EA
Indexed 2-Byte	5	3	DA

ROL (Rotate Left thru Carry)

Operation:



Description:

Shift all bits of the Accumulator, Index Register, or Memory one place to the left. Bit 0 is loaded from the Carry/Borrow bit. The Carry/Borrow bit is loaded from the most significant bit of the Accumulator, Index Register, or Memory.

Condition Codes:

H, 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 is set; cleared otherwise.

Boolean Formula(e) 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)

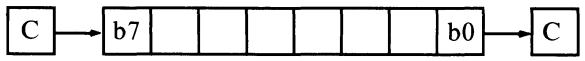
Source Form(s):

ROL Q, ROLA, ROLX

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	49
Index Register	3	1	59
Direct	5	2	39
Indexed 0 Offset	5	1	79
Indexed 1-Byte	6	2	69

ROR (Rotate Right Thru Carry)

Operation:



Description:

Shift all bits of the Accumulator, Index Register, or Memory one place to the right. Bit 7 is loaded from the Carry/Borrow bit. Bit 0 is loaded into the Carry/Borrow bit.

Condition Codes:

H, 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 is set; cleared otherwise.

ROR (Rotate Right Thru Carry) (Cont'd)

Boolean Formula(e) 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 = b0 (before operation)

Source Form(s):

ROR Q, RORA, RORX

Addressing Mode	Cycles	Bytes	Opcode
Accumulator	3	1	46
Index Register	3	1	56
Direct	5	2	36
Indexed 0 Offset	5	1	76
Indexed 1-Byte	6	2	66

RSP (Reset Stack Pointer)

Operation:

SP ← $7F

Description:

Reset the stack pointer to the top of the stack.

Condition Codes:

Not affected.

Source Form(s):

RSP

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	9C

RTI (Return from Interrupt)

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	Bytes	Opcode
Inherent	9	1	80

RTS (Return from Subroutine)

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	Bytes	Opcode
Inherent	6	1	81

SBC (Subtract with Carry)

Operation:
ACCA ← ACCA - M - C

Description:
Subtract the contents of Memory and the Carry/Borrow bit from the contents of the Accumulator, and place the result in the Accumulator.

Condition Codes:
H, 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 Formula(e) 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):
SBC P

Addressing Mode	Cycles	Bytes	Opcode
Immediate	2	2	A2
Direct	3	2	B2
Extended	4	3	C2
Indexed 0 Offset	3	1	F2
Indexed 1-Byte	4	2	E2
Indexed 2-Byte	5	3	D2

SEC (Set Carry Bit)

Operation:
C bit ← 1

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

Condition Codes:
H, I, N, Z: Not affected.
C: Set.

Boolean Formula(e) for Condition Codes:
C = 1

Source Form(s):
SEC

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	99

SEI (Set Interrupt Mask Bit)

Operation:
I bit ← 1

Description:
Set 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, N, Z, C: Not affected.
I: Set.

Boolean Formula(e) for Condition Codes:
I = 1

Source Form(s):
SEI

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	9B

STA (Store Accumulator in Memory)

Operation:
M ← ACCA

Description:
Store the contents of the Accumulator in Memory. The contents of the Accumulator remain the same.

Condition Codes:
H, I, C: 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.

Boolean Formula(e) for Condition Codes:
N = 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	Bytes	Opcode
Direct	4	2	B7
Extended	5	3	C7
Indexed 0 Offset	4	1	F7
Indexed 1-Byte	5	2	E7
Indexed 2-Byte	6	3	D7

STOP (Enable $\overline{IRQ}$, Stop Oscillator)

Description:
Reduce power consumption by eliminating all dynamic power dissipation, resulting in: (1) external interrupt request enabling, (2) inhibiting of oscillator, and, in the CDP6805E2/E3/F2/G2, (3) timer prescaler to clear, (4) disabling of timer interrupts, and (5) timer interrupt flag bit to clear. When $\overline{RESET}$ or $\overline{IRQ}$ input goes low: (1) oscillator is enabled, (2) a delay of some number of 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, N, Z, C: Not affected.
I: Cleared.

Source Form(s):
STOP

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	8E

STX (Store Index Register in Memory)

Operation:
M ← X

Description:
Store the contents of the Index Register in Memory. The contents of the Index Register remain the same.

Condition Codes:
H, I, C: 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.

Boolean Formula(e) 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	Bytes	Opcode
Direct	4	2	BF
Extended	5	3	CF
Indexed 0 Offset	4	1	FF
Indexed 1-Byte	5	2	EF
Indexed 2-Byte	6	3	DF

SUB (Subtract)

Operation:
ACCA ← ACCA - M

Description:
Subtract the contents of Memory from the contents of the Accumulator and place the result in the Accumulator.

Condition Codes:
H, 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 are larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formula(e) for Condition Codes:
$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$
$C = \overline{A7} \cdot M7 \vee R7 \cdot R7 \vee R7 \cdot \overline{A7}$

Source Form(s):
SUB P

Addressing Mode	Cycles	Bytes	Opcode
Immediate	2	2	A0
Direct	3	2	B0
Extended	4	3	C0
Indexed 0 Offset	3	1	F0
Indexed 1-Byte	4	2	E0
Indexed 2-Byte	5	3	D0

SWI (Software Interrupt)

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

SWI (Software Interrupt)

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

Boolean Formula(e) 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

Addressing Mode	Cycles	Bytes	Opcode
Inherent	10	1	83

TAX (Transfer Accumulator to Index Register)

Operation:
X ← ACCA

Description:
Load 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	Bytes	Opcode
Inherent	2	1	97

TST (Test for Negative or Zero)

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

Description:
Set the Negative Indicator and Zero Indicator condition code bits according to the contents of the Accumulator, Index Register, or Memory.

Condition Codes:
H, I, C: 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.

Boolean Formula(e) 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	Bytes	Opcode
Accumulator	3	1	4D
Index Register	3	1	5D
Direct	4	2	3D
Indexed 0 Offset	4	1	7D
Indexed 1-Byte	5	2	6D

WAIT (Enable Interrupt, Stop Processor)

Description:
Reduce power consumption by eliminating dynamic power dissipation in all circuits except the timer, serial peripheral interface, and serial communications interface. Enable external interrupts and stop clocking of processor circuits.

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

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

Interrupts are enabled following the RTI command.

Condition Codes:
H, N, Z, C: Not affected.
I: Cleared.

Source Form(s):
WAIT

Addressing Mode	Cycles	Bytes	Opcode
Inherent	2	1	8F

TXA (Transfer Index Register to Accumulator)

Operation:
ACCA ← X

Description:
Load 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	Bytes	Opcode
Inherent	2	1	9F

Appendix A
CDP6805 CMOS Family Compatibility with MC6800

Introduction

Strictly speaking, the CDP6805 CMOS Family is neither source-nor-object code compatible with the MC6800; but it is very similar to all 6800 Family processors. An experienced MC6800 programmer should have little difficulty adapting to the CDP6805 CMOS Family instruction set. The following paragraphs enumerate the differences between the MC6800 and the CDP6805 CMOS families.

Removed B-Register

In order to free valuable opcode space, the B-register is removed in the CDP6805 CMOS Family. Therefore, none of the register/memory or read/modify/write instructions have a B-register form. Several other instructions are also not available in the CDP6805 CMOS Family, including:

SBA, CBA, TAB, TBA, ABA, PSHB, and PULB

Removed V-Flag

The V-flag bit and the logic to set it are removed in the CDP6805 CMOS Family. This was done because usage of the small controller does not generally require signed arithmetic operations. However, unsigned arithmetic operations are still available. Without the V-flag bit, the following MC6800 instructions are not available in the CDP6805 CMOS Family:

SEV, CLV, BVC, BVS, BGE, BLT, BGT, and BLE

In the CDP6805 CMOS Family, unsigned inequalities are still available using BHS (BCC) and BLO (BCS).

Reduced Stack Control

Instructions relating to the manipulation of the SP are greatly reduced in the CDP6805 CMOS Family. On reset, or upon execution of the RSP instruction, the SP is initialized to $7F for the CDP6805E2/E3/F2/G2 and $FF for the CDP68HC05C4/D2. Other instructions that were deleted include:

LDS, STS, INS, DES, PSHA, PULA,
TXS, TSX, AND WAI

Do not confuse the WAIT instruction used with the CDP6805 CMOS Family with the WAI instruction used by the MC6800.

Removed DAA

The DAA has been deleted in the CDP6805 CMOS Family members. The H-bit, however, is retained and two additional branches are added to branch if the H-bit is set or cleared (BHCS, BHCC). These branches can be used to write software subroutines accomplishing DAA. (Remember, ROM is much cheaper than the DAA.)

Changed Register Lengths

The X-register is reduced to eight bits, the SP to eight bits or less, and the PC to 16 bits or less in the CDP6805 CMOS Family. The change in the X-register size from 16 to eight bits required changes in the addressing modes; these are described in the Addressing Modes paragraph of the section on **Software Description.** Also, because the X-and A-registers are equivalent in size, two new instructions are added to transfer X to A and A to X (TXA, TAX).

Bit Manipulation

Bit-manipulation instructions have been added to the CDP6805 CMOS Family because they are extremely useful for low-end applications. Two classes of bit-manipulation instructions have been added: bit set/clear and test-and-branch on bit set/clear.

(a) Bit Set/Clear

These instructions allow any bit in page zero, including bits in the I/O ports (but not always the data direction registers), to be set or cleared with one 2-byte instruction. Page zero includes the first 256 addressable memory locations from $00 through $FF.

(b) Test-and-Branch on Bit Set/Clear

These instructions test any bit in page zero (including I/O, RAM, and ROM) and will cause a branch, if the bit is set or cleared. In addition, the C-bit of the condition code register contains the state of the bit tested.

New Branches

Several new branches are added to facilitate low-end type programs in the CDP6805 CMOS Family. The BHCS and BHCC are useful in BCD additions. A branch, if the interupt mask bit is set or cleared (BMS/BMC), is also added. This eliminates the need for TAP and TPA because each bit in the condition code register can be tested by a branch. Two more branches are added that branch on the logic condition of the interrupt line (high or low): BIH/BIL. These allow the interrupt line to be used as an additional input in systems not using interrupts.

New Addressing Modes

The addressing modes of the MC6800 were optimized for the CDP6805 CMOS Family. For more details see the Addressing Modes paragraph in the section on **Software Description** of this manual.

Read/Modify/Write the X Register

By utilizing the column in the opcode map vacated by the B-register for read/modify/write, and because the X-register is now eight bits, all of these operations are available to the X-register. For example:

ROLX, INCX, CLRX, NEGX, etc.

This eliminated the traditional INX, DEX. However, mnemonics INX and DEX are still recognized by the assembler for compatibility.

Convenience Mnemonics

These are not new CDP6805 CMOS Family instructions, but only represent improvements to the M6805 HMOS/M146805 CMOS assembler that allow existing instructions to be recognized by more than one mnemonic.

(a) LSL (Logical Shift Left)

Because logical and arithmetic left shifts are identical, LSL is equivalent to ASL.

(b) BHS (Branch Higher or Same)

After a compare or subtract, the carry is cleared if the register argument was higher or equal to the memory argument; hence, the BHS is equivalent to BCC.

(c) BLO (Branch if Lower)

After a compare or subtract, the carry is set if the register argument was lower than the memory argument; hence, the BLO is equivalent to BCS.

Appendix B
Instruction Set
Alphabetical Listing

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

TABLE VI

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
ADC		X	X	X		X	X	X			^	•	^	^	^
ADD		X	X	X		X	X	X			^	•	^	^	^
AND		X	X	X		X	X	X			•	•	^	^	•
ASL	X		X			X	X				•	•	^	^	^
ASR	X		X			X	X				•	•	^	^	^
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			•	•	^	^	•
BLO					X						•	•	•	•	•
BLS					X						•	•	•	•	•
BMC					X						•	•	•	•	•
BMI					X						•	•	•	•	•
BMS					X						•	•	•	•	•
BNE					X						•	•	•	•	•
BPL					X						•	•	•	•	•
BRA					X						•	•	•	•	•
BRN					X						•	•	•	•	•
BRCLR										X	•	•	•	•	^
BRSET										X	•	•	•	•	^
BSET									X		•	•	•	•	•
BSR					X						•	•	•	•	•
CLC	X										•	•	•	•	0
CLI	X										•	0	•	•	•
CLR	X		X			X	X				•	•	0	1	•
CMP		X	X	X		X	X	X			•	•	^	^	^
COM	X		X			X	X				•	•	^	^	1
CPX		X	X	X		X	X	X			•	•	^	^	^

TABLE VI (Cont'd)

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				•	•	^	^	•
EOR		X	X	X		X	X	X			•	•	^	^	•
INC	X		X			X	X				•	•	^	^	•
JMP			X	X		X	X	X			•	•	•	•	•
JSR			X	X		X	X	X			•	•	•	•	•
LDA		X	X	X		X	X	X			•	•	^	^	•
LDX		X	X	X		X	X	X			•	•	^	^	•
LSL	X		X			X	X				•	•	^	^	^
LSR	X		X			X	X				•	•	0	^	^
MUL*	X										0	•	•	•	0
NEG	X		X			X	X				•	•	^	^	^
NOP	X										•	•	•	•	•
ORA		X	X	X		X	X	X			•	•	^	^	•
ROL	X		X			X	X				•	•	^	^	^
ROR	X		X			X	X				•	•	^	^	^
RSP	X										•	•	•	•	•
RTI	X										?	?	?	?	?
RTS	X										•	•	•	•	•
SBC		X	X	X		X	X	X			•	•	^	^	^
SEC	X										•	•	•	•	1
SEI	X										•	1	•	•	•
STA			X	X		X	X	X			•	•	^	^	•
STOP	X										•	1	•	•	•
STX			X	X		X	X	X			•	•	^	^	•
SUB		X	X	X		X	X	X			•	•	^	^	^
SWI	X										•	1	•	•	•
TAX	X										•	•	•	•	•
TST	X		X			X	X				•	•	^	^	•
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
- ^ Test and Set if True; Cleared Otherwise
- • Not Affected
- ? Load CC Register From Stack
- 0 Cleared
- 1 Set

*Note that the MUL instruction is available only on the CDP68HC05C4 and CDP68HC05D2 Microcomputers.

Appendix C
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) opcode, (4) number of bytes, and (5) number of cycles.

Table VII - Branch Instructions

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

Table VIII - Bit Manipulation Instructions

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

Table IX - Control Instructions

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

Table X - Read/Modify/Write Instructions

		Addressing Modes														
		Inherent (A)			Inherent (X)			Direct			Indexed (No Offset)			Indexed (8-Bit Offset)		
Function	Mnemonic	Op Code	# Bytes	# Cycles	Op Code	# Bytes	# Cycles	Op Code	# Bytes	# Cycles	Op Code	# Bytes	# Cycles	Op Code	# Bytes	# Cycles
Increment	INC	4C	1	3	5C	1	3	3C	2	5	7C	1	5	6C	2	6
Decrement	DEC	4A	1	3	5A	1	3	3A	2	5	7A	1	5	6A	2	6
Clear	CLR	4F	1	3	5F	1	3	3F	2	5	7F	1	5	6F	2	6
Complement	COM	43	1	3	53	1	3	33	2	5	73	1	5	63	2	6
Negate (2's Complement)	NEG	40	1	3	50	1	3	30	2	5	70	1	5	60	2	6
Rotate Left Thru Carry	ROL	49	1	3	59	1	3	39	2	5	79	1	5	69	2	6
Rotate Right Thru Carry	ROR	46	1	3	56	1	3	36	2	5	76	1	5	66	2	6
Logical Shift Left	LSL	48	1	3	58	1	3	38	2	5	78	1	5	68	2	6
Logical Shift Right	LSR	44	1	3	54	1	3	34	2	5	74	1	5	64	2	6
Arithmetic Shift Right	ASR	47	1	3	57	1	3	37	2	5	77	1	5	67	2	6
Test for Negative or Zero	TST	4D	1	3	5D	1	3	3D	2	4	7D	1	4	6D	2	5
Multiply	MUL	42	1	11	—	—	—	—	—	—	—	—	—	—	—	—

Table XI - Register/Memory Instructions

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

Appendix D
Instruction Set
Numerical Listing

This appendix provides a numerical listing of the operation codes used with the CDP6805 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.

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 CODE	MNEMONIC	MODE	# CYCLES	# BYTES
00	BRSET0	BTB	5	3
01	BRCLR0	BTB	5	3
02	BRSET1	BTB	5	3
03	BRCLR1	BTB	5	3
04	BRSET2	BTB	5	3
05	BRCLR2	BTB	5	3
06	BRSET3	BTB	5	3
07	BRCLR3	BTB	5	3
08	BRSET4	BTB	5	3
09	BRCLR4	BTB	5	3
0A	BRSET5	BTB	5	3
0B	BRCLR5	BTB	5	3
0C	BRSET6	BTB	5	3
0D	BRCLR6	BTB	5	3
0E	BRSET7	BTB	5	3
0F	BRCLR7	BTB	5	3
10	BSET0	BSC	5	2
11	BCLR0	BSC	5	2
12	BSET1	BSC	5	2
13	BCLR1	BSC	5	2
14	BSET2	BSC	5	2
15	BCLR2	BSC	5	2
16	BSET3	BSC	5	2
17	BCLR3	BSC	5	2
18	BSET4	BSC	5	2
19	BCLR4	BSC	5	2
1A	BSET5	BSC	5	2
1B	BCLR5	BSC	5	2
1C	BSET6	BSC	5	2
1D	BCLR6	BSC	5	2
1E	BSET7	BSC	5	2
1F	BCLR7	BSC	5	2
20	BRA	REL	3	2
21	BRN	REL	3	2
22	BHI	REL	3	2
23	BLS	REL	3	2
24	BCC	REL	3	2
25	BCS	REL	3	2
26	BNE	REL	3	2
27	BEQ	REL	3	2
28	BHCC	REL	3	2
29	BHCS	REL	3	2
2A	BPL	REL	3	2
2B	BMI	REL	3	2
2C	BMC	REL	3	2
2D	BMS	REL	3	2
2E	BIL	REL	3	2
2F	BIH	REL	3	2
30	NEG	DIR	5	2
33	COM	DIR	5	2
34	LSR	DIR	5	2
36	ROR	DIR	5	2
37	ASR	DIR	5	2

INSTRUCTION SET NUMERICAL LISTING (CONTINUED)

OP CODE	MNEMONIC	MODE	# CYCLES	# BYTES
38	LSL	DIR	5	2
39	ROL	DIR	5	2
3A	DEC	DIR	5	2
3C	INC	DIR	5	2
3D	TST	DIR	4	2
3F	CLR	DIR	5	2
40	NEGA	INH	3	1
42	MUL*	INH	11	1
43	COMA	INH	3	1
44	LSRA	INH	3	1
46	RORA	INH	3	1
47	ASRA	INH	3	1
48	LSLA	INH	3	1
49	ROLA	INH	3	1
4A	DECA	INH	3	1
4C	INCA	INH	3	1
4D	TSTA	INH	3	1
4F	CLRA	INH	3	1
50	NEGX	INH	3	1
53	COMX	INH	3	1
54	LSRX	INH	3	1
56	RORX	INH	3	1
57	ASRX	INH	3	1
58	LSLX	INH	3	1
59	ROLX	INH	3	1
5A	DECX	INH	3	1
5C	INCX	INH	3	1
5D	TSTX	INH	3	1
5F	CLRX	INH	3	1
60	NEG	IX1	6	2
63	COM	IX1	6	2
64	LSR	IX1	6	2
66	ROR	IX1	6	2
67	ASR	IX1	6	2
68	LSL	IX1	6	2
69	ROL	IX1	6	2
6A	DEC	IX1	6	2
6C	INC	IX1	6	2
6D	TST	IX1	5	2
6F	CLR	IX1	6	2
70	NEG	IX	5	1
73	COM	IX	5	1
74	LSR	IX	5	1
76	ROR	IX	5	1
77	ASR	IX	5	1
78	LSL	IX	5	1
79	ROL	IX	5	1
7A	DEC	IX	5	1
7C	INC	IX	5	1
7D	TST	IX	4	1
7F	CLR	IX	5	1
80	RTI	INH	9	1
81	RTS	INH	6	1
83	SWI	INH	10	1

INSTRUCTION SET NUMERICAL LISTING (CONTINUED)

OP CODE	MNEMONIC	MODE	# CYCLES	# BYTES
8E	STOP	INH	2	1
8F	WAIT	INH	2	1
97	TAX	INH	2	1
98	CLC	INH	2	1
99	SEC	INH	2	1
9A	CLI	INH	2	1
9B	SEI	INH	2	1
9C	RSF	INH	2	1
9D	NOP	INH	2	1
9F	TXA	INH	2	1
A0	SUB	IMM	2	2
A1	CMP	IMM	2	2
A2	SBC	IMM	2	2
A3	CPX	IMM	2	2
A4	AND	IMM	2	2
A5	BIT	IMM	2	2
A6	LDA	IMM	2	2
A8	EOR	IMM	2	2
A9	ADC	IMM	2	2
AA	ORA	IMM	2	2
AB	ADD	IMM	2	2
AD	BSR	IMM	6	2
AE	LDX	IMM	2	2
B0	SUB	DIR	3	2
B1	CMP	DIR	3	2
B2	SBC	DIR	3	2
B3	CPX	DIR	3	2
B4	AND	DIR	3	2
B5	BIT	DIR	3	2
B6	LDA	DIR	3	2
B7	STA	DIR	4	2
B8	EOR	DIR	3	2
B9	ADC	DIR	3	2
BA	ORA	DIR	3	2
BB	ADD	DIR	3	2
BC	JMP	DIR	3	2
BD	JSR	DIR	5	2
BE	LDX	DIR	3	2
BF	STX	DIR	4	2
C0	SUB	EXT	4	3
C1	CMP	EXT	4	3
C2	SBC	EXT	4	3
C3	CPX	EXT	4	3
C4	AND	EXT	4	3
C5	BIT	EXT	4	3
C6	LDA	EXT	4	3
C7	STA	EXT	5	3
C8	EOR	EXT	4	3
C9	ADC	EXT	4	3
CA	ORA	EXT	4	3
CB	ADD	EXT	4	3
CC	JMP	EXT	4	3
CD	JSR	EXT	6	3
CE	LDX	EXT	4	3
CF	STX	EXT	5	3

INSTRUCTION SET NUMERICAL LISTING (CONTINUED)

OP CODE	MNEMONIC	MODE	# CYCLES	# BYTES
D0	SUB	IX2	5	3
D1	CMP	IX2	5	3
D2	SBC	IX2	5	3
D3	CPX	IX2	5	3
D4	AND	IX2	5	3
D5	BIT	IX2	5	3
D6	LDA	IX2	5	3
D7	STA	IX2	6	3
D8	EOR	IX2	5	3
D9	ADC	IX2	5	3
DA	ORA	IX2	5	3
DB	ADD	IX2	5	3
DC	JMP	IX2	5	3
DD	JSR	IX2	7	3
DE	LDX	IX2	5	3
DF	STX	IX2	6	3
E0	SUB	IX1	4	2
E1	CMP	IX1	4	2
E2	SBC	IX1	4	2
E3	CPX	IX1	4	2
E4	AND	IX1	4	2
E5	BIT	IX1	4	2
E6	LDA	IX1	4	2
E7	STA	IX1	5	2
E8	EOR	IX1	4	2
E9	ADC	IX1	4	2
EA	ORA	IX1	4	2
EB	ADD	IX1	4	2
EC	JMP	IX1	4	2
ED	JSR	IX1	6	2
EE	LDX	IX1	4	2
EF	STX	IX1	5	2
F0	SUB	IX	3	1
F1	CMP	IX	3	1
F2	SBC	IX	3	1
F3	CPX	IX	3	1
F4	AND	IX	3	1
F5	BIT	IX	3	1
F6	LDA	IX	3	1
F7	STA	IX	4	1
F8	EOR	IX	3	1
F9	ADC	IX	3	1
FA	ORA	IX	3	1
FB	ADD	IX	3	1
FC	JMP	IX	3	1
FD	JSR	IX	5	1
FE	LDX	IX	3	1
FF	STX	IX	4	1

*Note that the MUL (42) instruction is available only on the CDP68HC05C4 and CDP68HC05D2 Microcomputers.

Appendix E
Instruction Set Cycle-By-Cycle Operation Summary

Table XII provides a detailed description of the cycle-by-cycle operation for each instruction in the CDP6805 CMOS Family. The table contains information which includes the total number of cycles required to execute the instruction, plus a step-by-step breakdown of each cycle. Except for the CDP6805E2 Microprocessor Unit (MPU), all of the CDP6805 CMOS Family members are Microcomputer Units (MCUs). This means that only the CDP6805E2 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 this table 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 XII - CDP6805 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	2	1	Opcode Address	1	Opcode
SEC SEI TAX TXA		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 XII - CDP6805 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	Opcod 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 XII - CDP6805 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	3	1	Opcode Address	1	Opcode
CMP CPX EOR LDA		2	Opcode Address + 1	1	Opcode Next Instruction
LDX ORA SBC SUB		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	4	1	Opcode Address	1	Opcode
STX		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	5	1	Opcode Address	1	Opcode
COM DEC INC		2	Opcode Address + 1	1	Opcode Next Instruction
LSL LSR NEG		3	Index Register	1	Current Operand Data
ROL ROR		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	4	1	Opcode Address	1	Opcode
BIT CMP CPX		2	Opcode Address + 1	1	Offset
EOR LDA LDX		3	Opcode Address + 1	1	Offset
ORA SBC SUB		4	Index Register + Offset	1	Operand Data
STA	5	1	Opcode Address	1	Opcode
STX		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	6	1	Opcode Address	1	Opcode
COM DEC INC		2	Opcode Address + 1	1	Offset
LSL LSR NEG		3	Opcode Address + 1	1	Offset
ROL ROR		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 XII - CDP6805 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 XII - CDP6805 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 CDP6805E2 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., CDP6805E2 and CDP6805G2). For devices with 11-bit program counters (CDP6805F2), the HI byte is "07" instead of "1F".

X indicates don't care.

***On the CDP6805E2 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.

Appendix F
Instruction Set
Opcode Map

The opcode map contains a summary of opcodes used with the CDP6805 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 the CDP6805 CMOS Family, the required number of bytes, the address mode, and the mnemonic.

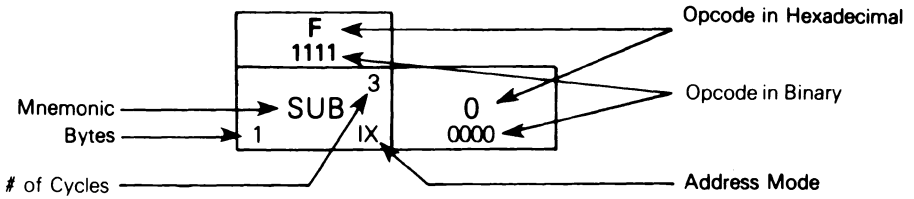
TABLE XIII - Instruction Set Opcode Map

	Bit Manipulation		Branch	Read/Modify/Write					Control		Register/Memory						
	BTB	BSC	REL	DIR	INH(A)	INH(X)	IX1	IX	INH	INH	IMM	DIR	EXT	IX2	IX1	IX	
Hi / Low	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	Hi / Low
0 0000	5 BRSET0 3 BTB	5 BSET0 2 BSC	3 BRA 2 REL	5 NEG 2 DIR	3 NEGA 1 INH	3 NEGX 1 INH	6 NEG 2 IX1	5 NEG 1 IX	9 RTI 1 INH		2 SUB 2 IMM	3 SUB 2 DIR	4 SUB 3 EXT	5 SUB 3 IX2	4 SUB 2 IX1	3 SUB 1 IX	0 0000
1 0001	5 BRCLR0 3 BTB	5 BCLR0 2 BSC	3 BRN 2 REL						6 RTS 1 INH		2 CMP 2 IMM	3 CMP 2 DIR	4 CMP 3 EXT	5 CMP 3 IX2	4 CMP 2 IX1	3 CMP 1 IX	1 0001
2 0010	5 BRSET1 3 BTB	5 BSET1 2 BSC	3 BHI 2 REL		11 MUL* 1 INH						2 SBC 2 IMM	3 SBC 2 DIR	4 SBC 3 EXT	5 SBC 3 IX2	4 SBC 2 IX1	3 SBC 1 IX	2 0010
3 0011	5 BRCLR1 3 BTB	5 BCLR1 2 BSC	3 BLS 2 REL	5 COM 2 DIR	3 COMA 1 INH	3 COMX 1 INH	6 COM 2 IX1	5 COM 1 IX	10 SWI 1 INH		2 CPX 2 IMM	3 CPX 2 DIR	4 CPX 3 EXT	5 CPX 3 IX2	4 CPX 2 IX1	3 CPX 1 IX	3 0011
4 0100	5 BRSET2 3 BTB	5 BSET2 2 BSC	3 BCC 2 REL	5 LSR 2 DIR	3 LSRA 1 INH	3 LSRX 1 INH	6 LSR 2 IX1	5 LSR 1 IX			2 AND 2 IMM	3 AND 2 DIR	4 AND 3 EXT	5 AND 3 IX2	4 AND 2 IX1	3 AND 1 IX	4 0100
5 0101	5 BRCLR2 3 BTB	5 BCLR2 2 BSC	3 BCS 2 REL								2 BIT 2 IMM	3 BIT 2 DIR	4 BIT 3 EXT	5 BIT 3 IX2	4 BIT 2 IX1	3 BIT 1 IX	5 0101
6 0110	5 BRSET3 3 BTB	5 BSET3 2 BSC	3 BNE 2 REL	5 ROR 2 DIR	3 RORA 1 INH	3 RORX 1 INH	6 ROR 2 IX1	5 ROR 1 IX			2 LDA 2 IMM	3 LDA 2 DIR	4 LDA 3 EXT	5 LDA 3 IX2	4 LDA 2 IX1	3 LDA 1 IX	6 0110
7 0111	5 BRCLR3 3 BTB	5 BCLR3 2 BSC	3 BEQ 2 REL	5 ASR 2 DIR	3 ASRA 1 INH	3 ASRX 1 INH	6 ASR 2 IX1	5 ASR 1 IX		2 TAX 1 INH		4 STA 2 DIR	5 STA 3 EXT	6 STA 3 IX2	5 STA 2 IX1	4 STA 1 IX	7 0111
8 1000	5 BRSET4 3 BTB	5 BSET4 2 BSC	3 BHCC 2 REL	5 LSL 2 DIR	3 LSLA 1 INH	3 LSLX 1 INH	6 LSL 2 IX1	5 LSL 1 IX		2 CLC 1 INH	2 EOR 2 IMM	3 EOR 2 DIR	4 EOR 3 EXT	5 EOR 3 IX2	4 EOR 2 IX1	3 EOR 1 IX	8 1000
9 1001	5 BRCLR4 3 BTB	5 BCLR4 2 BSC	3 BHCS 2 REL	5 ROL 2 DIR	3 ROLA 1 INH	3 ROLX 1 INH	6 ROL 2 IX1	5 ROL 1 IX		2 SEC 1 INH	2 ADC 2 IMM	3 ADC 2 DIR	4 ADC 3 EXT	5 ADC 3 IX2	4 ADC 2 IX1	3 ADC 1 IX	9 1001
A 1010	5 BRSET5 3 BTB	5 BSET5 2 BSC	3 BPL 2 REL	5 DEC 2 DIR	3 DECA 1 INH	3 DECX 1 INH	6 DEC 2 IX1	5 DEC 1 IX		2 CLI 1 INH	2 ORA 2 IMM	3 ORA 2 DIR	4 ORA 3 EXT	5 ORA 3 IX2	4 ORA 2 IX1	3 ORA 1 IX	A 1010
B 1011	5 BRCLR5 3 BTB	5 BCLR5 2 BSC	3 BMI 2 REL							2 SEI 1 INH	2 ADD 2 IMM	3 ADD 2 DIR	4 ADD 3 EXT	5 ADD 3 IX2	4 ADD 2 IX1	3 ADD 1 IX	B 1011
C 1100	5 BRSET6 3 BTB	5 BSET6 2 BSC	3 BMC 2 REL	5 INC 2 DIR	3 INCA 1 INH	3 INCX 1 INH	6 INC 2 IX1	5 INC 1 IX		2 RSP 1 INH		2 JMP 2 DIR	3 JMP 3 EXT	4 JMP 3 IX2	3 JMP 2 IX1	2 JMP 1 IX	C 1100
D 1101	5 BRCLR6 3 BTB	5 BCLR6 2 BSC	3 BMS 2 REL	4 TST 2 DIR	3 TSTA 1 INH	3 TSTX 1 INH	5 TST 2 IX1	4 TST 1 IX		2 NOP 1 INH	6 BSR 2 REL	5 JSR 2 DIR	6 JSR 3 EXT	7 JSR 3 IX2	6 JSR 2 IX1	5 JSR 1 IX	D 1101
E 1110	5 BRSET7 3 BTB	5 BSET7 2 BSC	3 BIL 2 REL						2 STOP 1 INH		2 LDX 2 IMM	3 LDX 2 DIR	4 LDX 3 EXT	5 LDX 3 IX2	4 LDX 2 IX1	3 LDX 1 IX	E 1110
F 1111	5 BRCLR7 3 BTB	5 BCLR7 2 BSC	3 BIH 2 REL	5 CLR 2 DIR	3 CLRA 1 INH	3 CLRX 1 INH	6 CLR 2 IX1	5 CLR 1 IX	2 WAIT 1 INH	2 TXA 1 INH		4 STX 2 DIR	5 STX 3 EXT	6 STX 3 IX2	5 STX 2 IX1	4 STX 1 IX	F 1111

Abbreviations for Address Modes

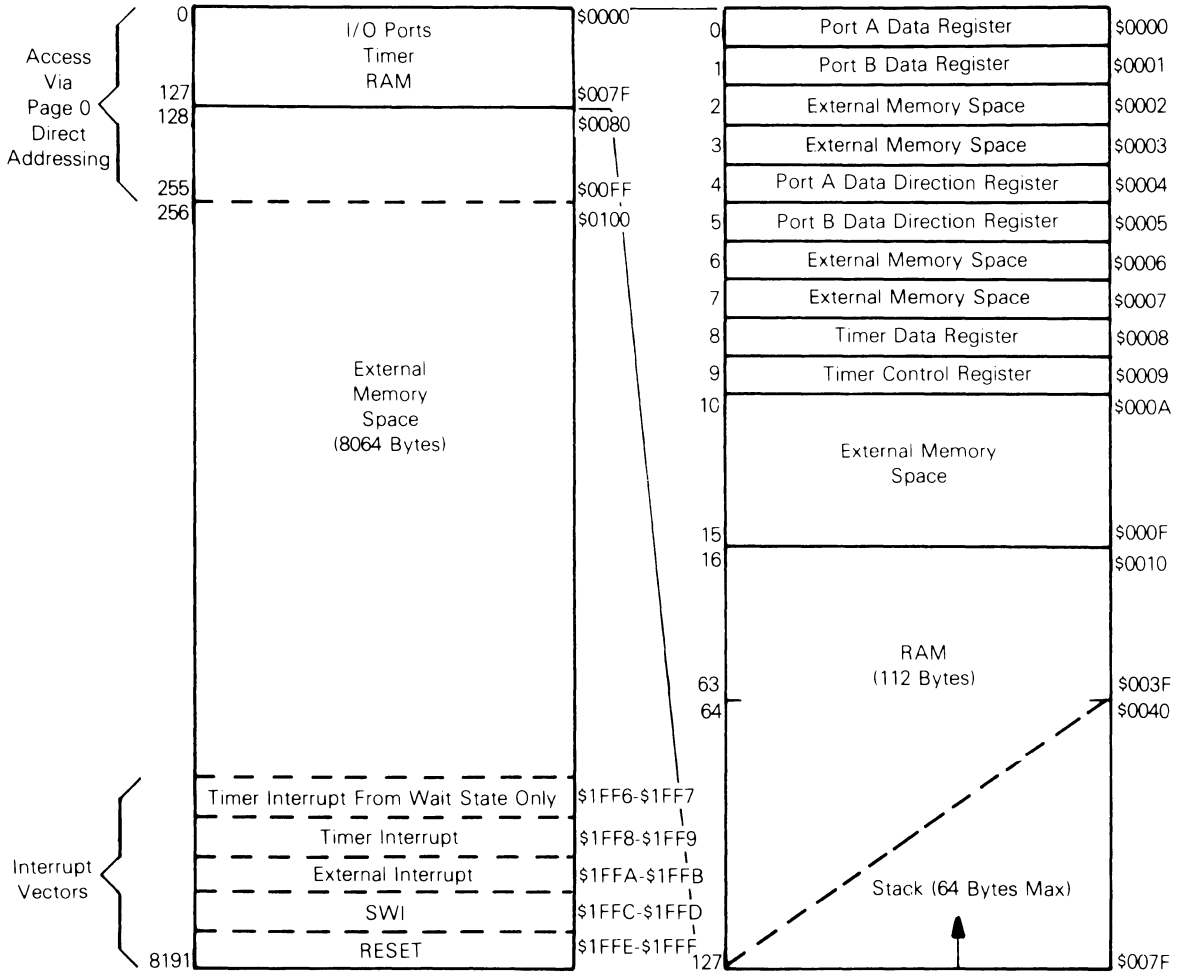
INH	Inherent	IX	Indexed (No Offset)
IMM	Immediate	IX1	Indexed, 1-Byte (8-Bit) Offset
DIR	Direct	IX2	Indexed, 2-Byte (16-Bit) Offset
EXT	Extended	*	Available only on CDP68HC05C4 and CDP68HC05D2
REL	Relative		
BSC	Bit Set/Clear	A	Accumulator
BTB	Bit Test and Branch	X	Index Register

LEGEND



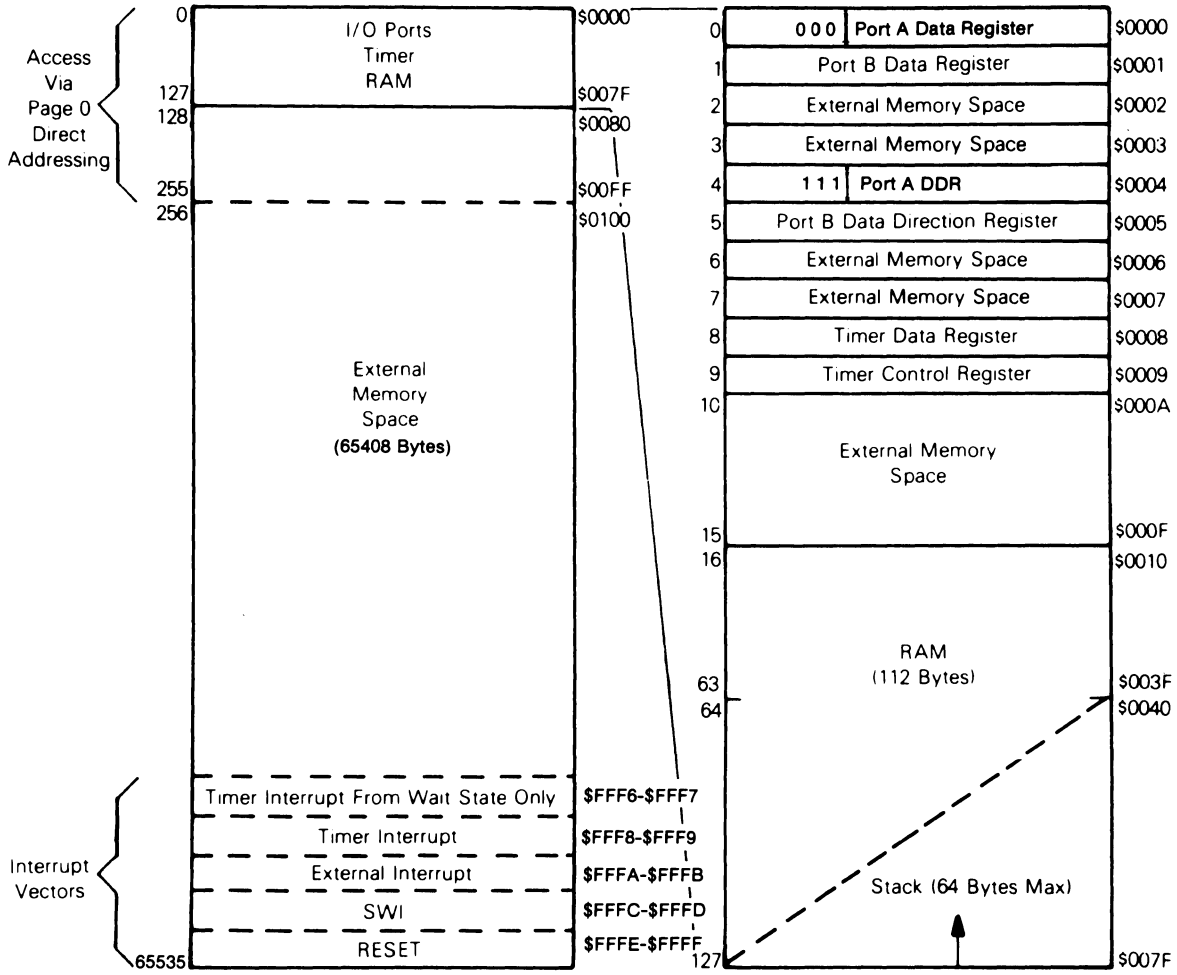
Appendix G
Address Maps for the CDP6805 CMOS Family

CDP6805E2 Address Map



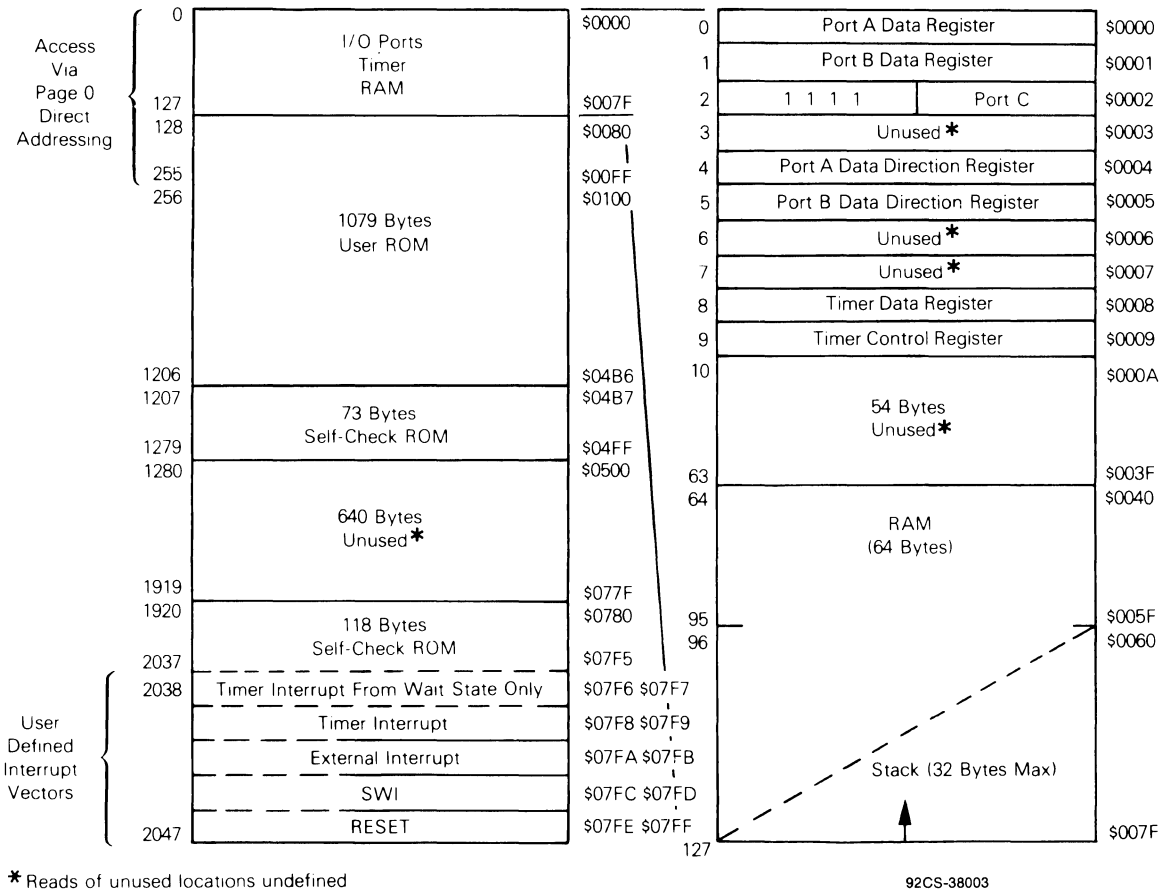
92CS-38027

CDP6805E3 Address Map



92CS-38359

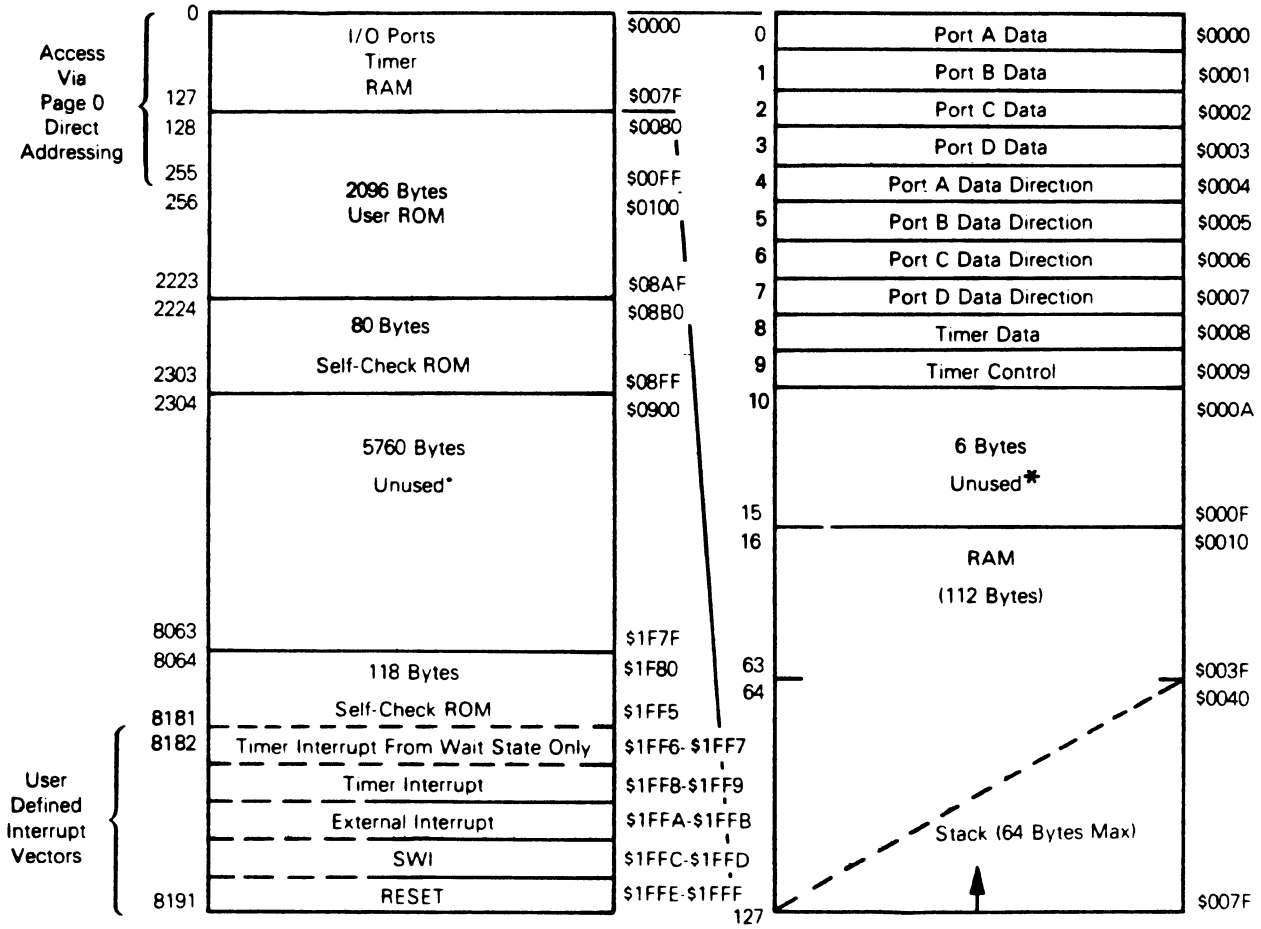
CDP6805F2 Address Map



* Reads of unused locations undefined

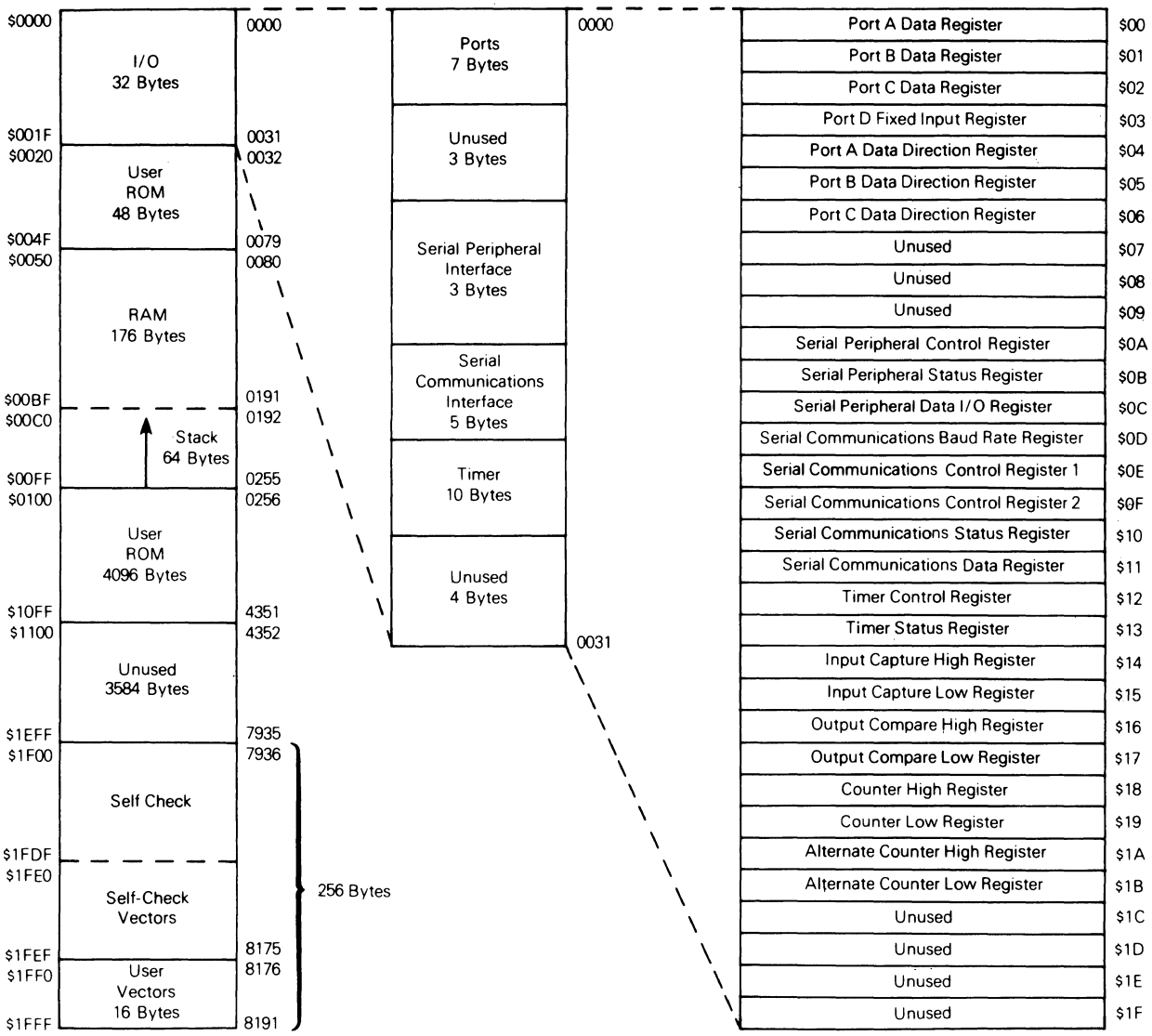
92CS-38003

CDP6805G2 Address Map

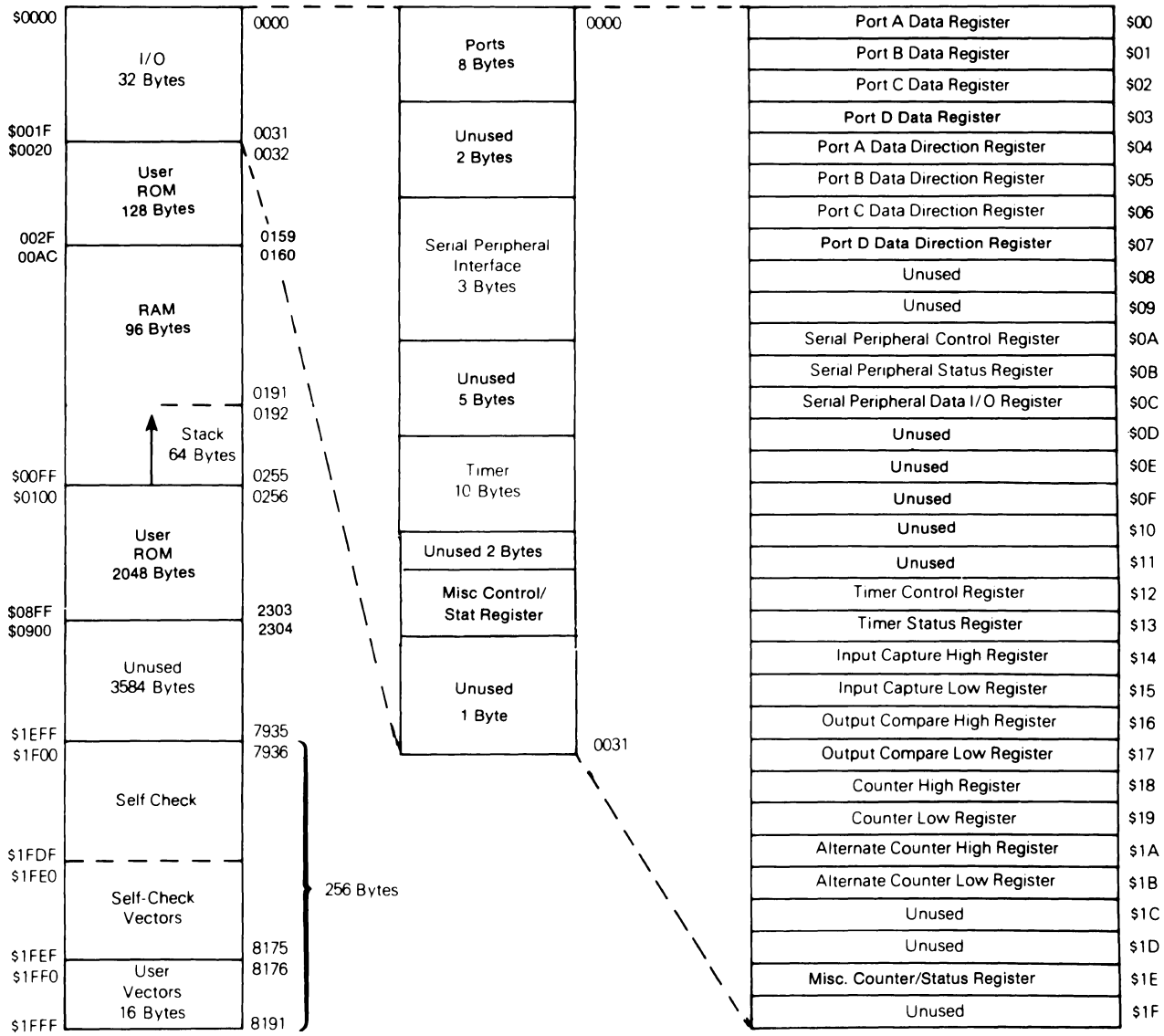


*Reads of unused locations undefined.

CDP68HC05C4 Address Map



CDP68HC05D2 Address Map



92CS-38118

Appendix H
ASCII Hexadecimal Code Conversion Chart

This appendix shows the equivalent alphanumeric characters for the equivalent ASCII hexadecimal code.

ASCII Character Set

MOST SIGNIFICANT CHARACTER

LEAST SIGNIFICANT CHARACTER HEX	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	—	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

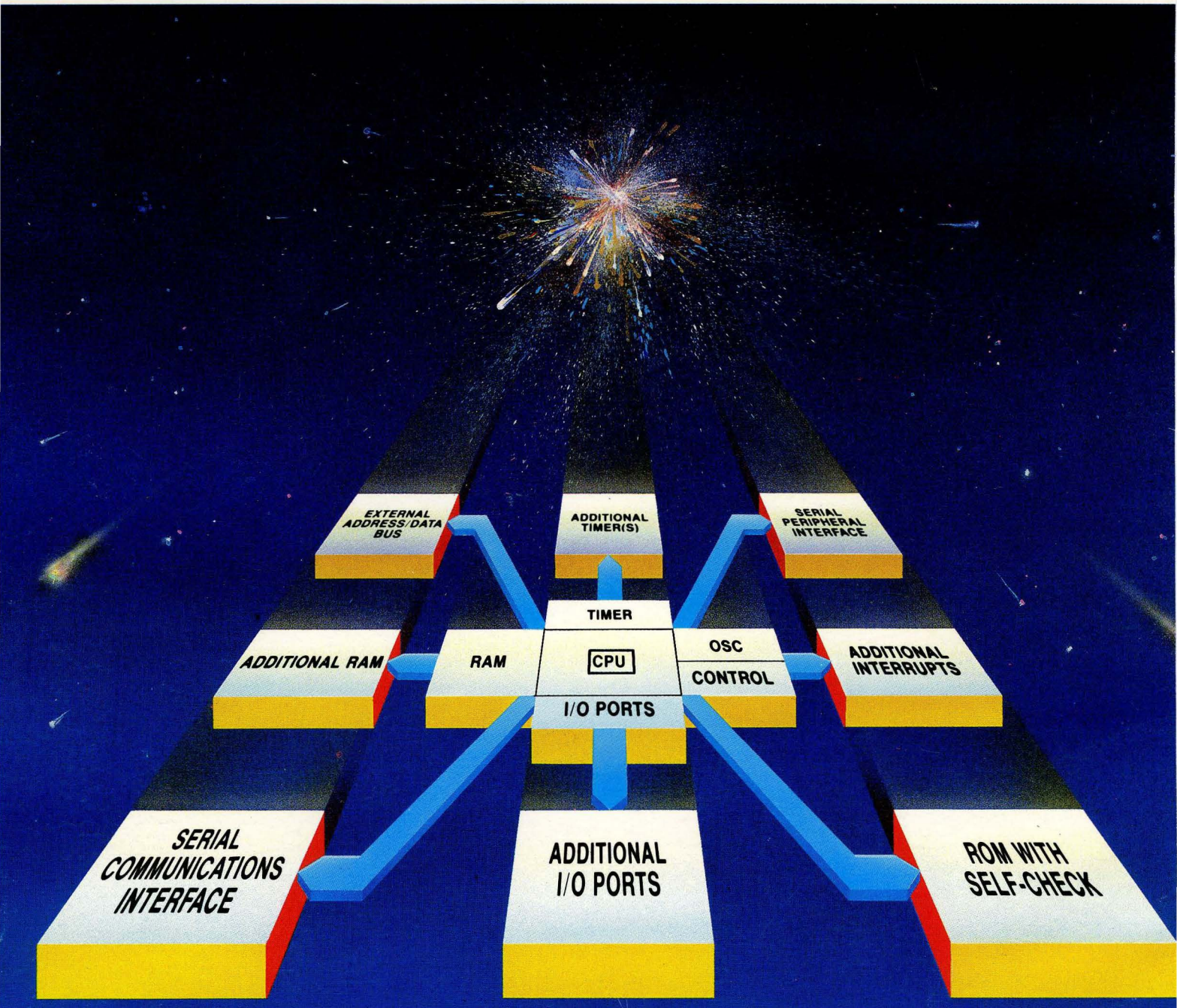
NOTES:
(1) Parity bit in most significant hex digit not included.
(2) Characters in columns 0 and 1 (as well as SP and DEL) are non-printing.
(3) Model 33 Teletypewriter prints codes in columns 6 and 7 as if they were column 4 and 5 codes.

92CS-34736R1

Index

RCA

User Manual



MPM-265