SECTION 3
MC68HC705C8 FUNCTIONAL DATA

The MC68HC705C8 microcontroller (MCU) is a member of the M68HC05 Family of low-cost, single-chip microcontrollers.

The HCMOS technology used on the MC68HC705C8 combines smaller size and higher speeds with the low power and high noise immunity of CMOS.

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

Two software-controlled power-saving modes, WAIT and STOP, are available to conserve additional power. These modes make the MC68HC705C8 especially attractive for automotive and battery-driven applications.

3

3.1 MCU DESCRIPTION

The hardware and software highlights of the MC68HC705C8 are as follows:

Hardware Features
- HCMOS Technology
- 8-Bit Architecture
- Power-Saving Stop, Wait, and Data Retention Modes
- 24 Bidirectional I/O Lines
- 7 Input-Only Lines
- 2 Timer I/O Pins
- 2.1 MHz Internal Operating Frequency, 5 Volts; 1.0 MHz, 3 Volts
- Internal 16-Bit Timer
- Serial Communications Interface (SCI) System
- Serial Peripheral Interface (SPI) System

■ 6367248 0149786 602 ■

- Ultraviolet (UV) light EPROM or One-Time Programmable ROM (OTPROM)
- Selectable Memory Configurations
- Computer Operating Properly (COP) Watchdog System
- Clock Monitor
- On-Chip Bootstrap Firmware for Programming
- Software-Programmable External Interrupt Sensitivity
- External Pin, Timer, SCI, and SPI Interrupts
- Master Reset and Power-On Reset
- Single 3- to 6-Volt Supply (2-Volt Data Retention Mode)
- On-Chip Oscillator
- 40-Pin Dual-in-Line Package or
- 44-Lead PLCC (Plastic Leaded Chip Carrier) Package

Software Features
- Upward Software Compatible with the M146805 CMOS Family
- Efficient Instruction Set
- Versatile Interrupt Handling
- True Bit Manipulation
- Addressing Modes with Indexed Addressing for Tables
- Memory-Mapped I/O
- Two Power-Saving Standby Modes

Figure 3-1 shows the MC68HC705C8 MCU block diagram.

The central processor unit (CPU) contains the 8-bit arithmetic logic unit, accumulator, index register, condition code register, stack pointer, program counter, and CPU control logic.

Major peripheral functions are provided on-chip. On-chip memory systems include bootstrap read-only memory (ROM), programmable ROM (EPROM or OTPROM), and random-access memory (RAM).

On-chip I/O devices include an asynchronous serial communications interface (SCI), a separate serial peripheral interface (SPI), and a 16-bit programmable timer system.

Self-monitoring circuitry is included on-chip to protect against system errors. A computer operating properly (COP) watchdog system protects against software failures. A clock monitor system generates a system reset if the clock is lost or runs too slow. An illegal opcode detection circuit provides a non-maskable interrupt if an illegal opcode is detected.
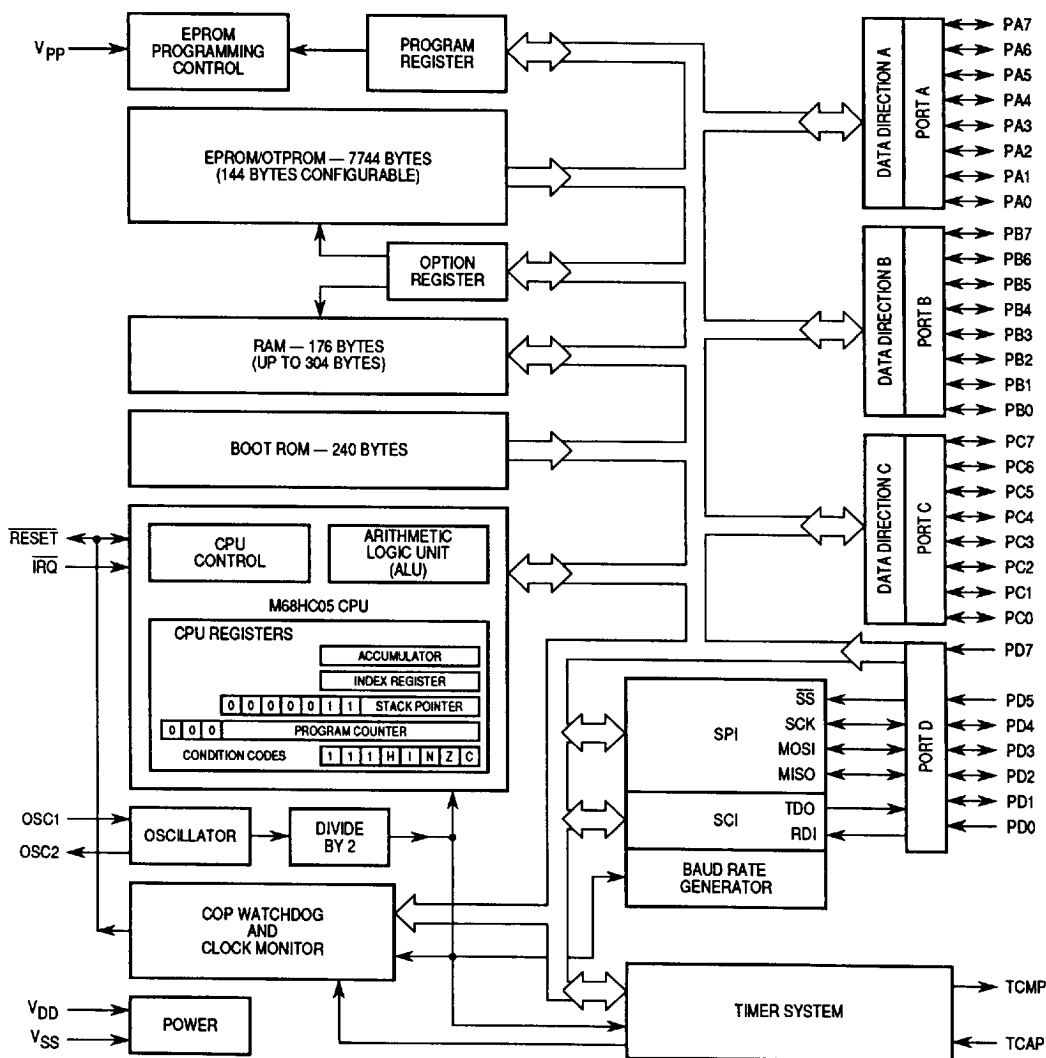
■ 6367248 0149787 549 ■

Figure 3-1. MC68HC705C8 Microcontroller Block Diagram

■ 6367248 0149788 485 ■

3.2 PINS AND CONNECTIONS

The following paragraphs discuss the MCU pin assignments, pin functions, and basic connections.

Because the MC68HC705C8 is a CMOS device, unused input pins must be terminated to avoid oscillation, noise, and added supply current. The preferred method of terminating pins that can be configured for input or output is with individual pullup or pulldown resistors for each unused pin.

Pin assignments are shown in Figure 3-2. Mechanical data and ordering information can be found in BR594/D, the *MC68HC705C8 Technical Summary*, available separately.

3.2.1 Pin Functions

V_{DD} and V_{SS}

Power is supplied to the MCU using these two pins. V_{DD} is power and V_{SS} is ground. The MCU can operate from a single 5-volt (nominal) power supply.

V_{PP}

The V_{PP} pin is used when programming the one-time programmable ROM (OTPROM) or EPROM. Programming voltage (14.75 Vdc) is applied to this pin when programming the PROM. Normally, this pin is connected to V_{DD}.

CAUTION

Do not connect V_{PP} pin to V_{SS} (GND). It will damage the MCU.

IRQ (Maskable Interrupt Request)

IRQ is a software programmable option which provides two different choices of interrupt triggering sensitivity. These options are 1) negative edge-sensitive triggering only, or 2) both negative edge-sensitive and level-sensitive triggering.

In the latter case, either a negative edge or a low level input to the IRQ pin will produce an interrupt. The MCU completes the current instruction before it responds to the interrupt request. When the IRQ pin goes low, a

■ 6367248 0149789 311 ■
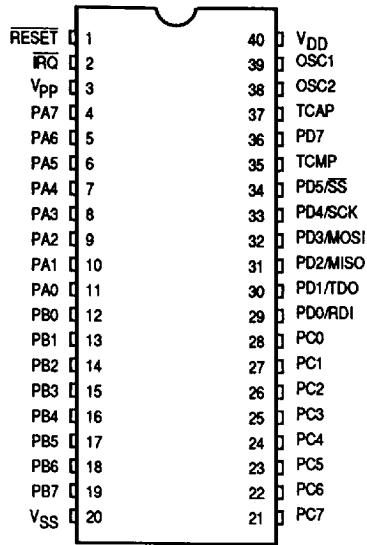
40-Pin Dual-In-Line Package
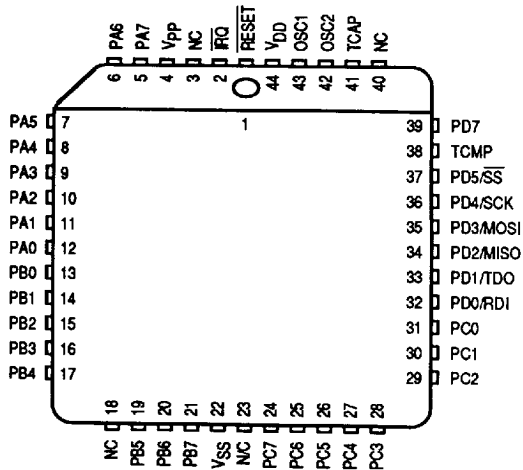
44-Lead PLCC Package

Figure 3-2. Pin Assignments

6367248 0149790 033

small synchronization delay occurs, and a logic one is latched internally to signify an interrupt has been requested. When the MCU completes its current instruction, the interrupt latch is tested. If the interrupt latch contains a logic one and the interrupt mask bit (I bit) in the condition code register is clear, the MCU then begins the interrupt sequence.

If the option is selected to include level-sensitive triggering, then the $\overline{\text{IRQ}}$ input requires an external resistor to V_{DD} for "wired-OR" operation. See **3.4.7 Interrupts** for more detail concerning interrupts.

$\overline{\text{RESET}}$

The $\overline{\text{RESET}}$ pin is an active-low bidirectional control signal. As an input, the $\overline{\text{RESET}}$ pin initializes the MCU to a known startup state. As an open-drain output, the $\overline{\text{RESET}}$ pin indicates an internal MCU failure detected by the computer operating properly (COP) watchdog timer or clock monitor circuitry.

This $\overline{\text{RESET}}$ pin is significantly different from the RESET signal used on other Motorola M68HC05 Family devices. Refer to **3.4.4 Resets** and **3.4.7 Interrupts** before designing circuitry to generate or monitor the RESET signal.

TCAP

The TCAP pin provides the input to the input-capture feature for the on-chip programmable timer system. Refer to input-capture register in **3.8 PROGRAMMABLE TIMER**.

TCMP

The TCMP pin provides an output for the output-compare feature of the on-chip timer system. Refer to output-compare register in **3.8 PROGRAMM-ABLE TIMER**.

OSC1,OSC2

The MC68HC705C8 can accept either a crystal, ceramic resonator, or external input to control the internal oscillator. The internal processor clock is derived by dividing the oscillator frequency (f_{osc}) by two.

The circuit shown in Figure 3-3(a) is recommended when using a crystal. The internal oscillator is designed to interface with an AT-cut parallel resonant quartz crystal or a ceramic resonator up to 4 MHz. The crystal and components should be mounted as close as possible to the input pins to minimize output distortion and startup stabilization time.

■ 6367248 0149791 T7T ■
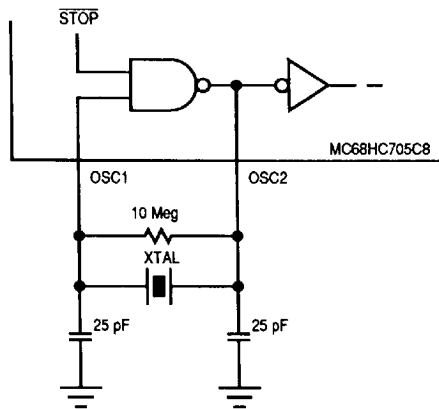
(a) Crystal/Ceramic Resonator Oscillator Connections
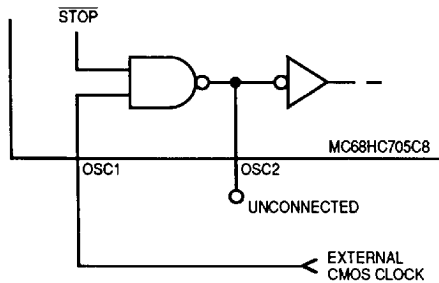
(b) External Clock Source Connections

Figure 3-3. Oscillator Connections

■ 6367248 0149792 906 ■

A ceramic resonator may be used in place of the crystal in cost-sensitive applications. The circuit in Figure 3-3(a) is recommended when using a ceramic resonator or a crystal. The manufacturer of the particular ceramic resonator being considered should be consulted for specific information.

An external clock may be applied to the OSC1 input with the OSC2 pin not connected, as shown in Figure 3-3(b).

PA7–PA0

These eight I/O lines comprise port A. Each port A pin can be software programmed to act as an input or output.

PB7–PB0

These eight lines comprise port B. Each port B pin can be software programmed to act as an input or output.

PC7–PC0

These eight lines comprise port C. Each port C pin can be software programmed to act as an input or output.

PD5–PD0, PD7

These seven lines comprise port D. During power-on or reset, these seven pins are configured as inputs. When the SPI system is enabled, four of these lines, MISO/PD2, MOSI/PD3, SCK/PD4, and SS/PD5, are used by the SPI system. When the SCI receiver is enabled, the PD0/RDI pin becomes the receive data input to the SCI. When the SCI transmitter is enabled, the PD1/TDO pin becomes the transmit data output for the SCI.

3.2.2 Typical Basic Connections

There are MCU basic connections that can be used as the starting point for any application to minimize the time required to create a prototype system.

Figure 3-4 is the schematic diagram for a simple MC68HC705C8 system. This circuit can be used as the basis for any MC68HC705C8 application. In most cases, the circuitry for the power supply and oscillator can be used as shown in this diagram. All unused inputs are terminated in an appropriate manner.
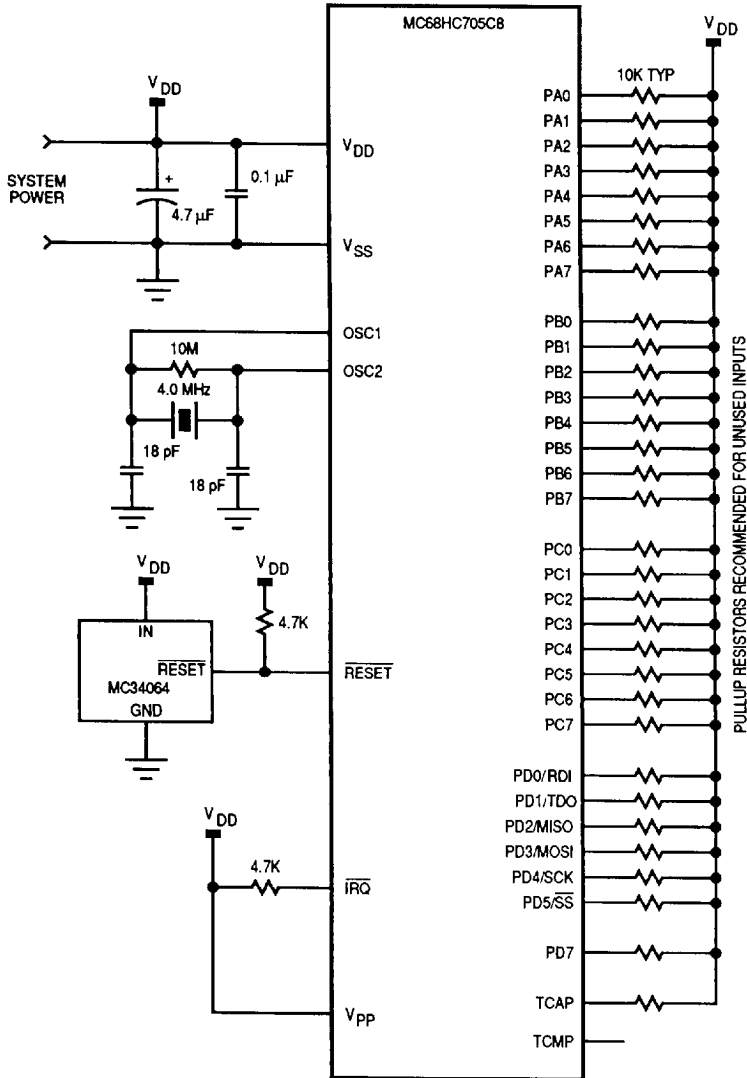
Figure 3-4. Typical Basic Connections

3.3 ON-CHIP MEMORY

The MC68HC705C8 memory includes 176 to 304 bytes of random-access memory (RAM), 240 bytes of read-only memory (ROM), and 7600 to 7744 bytes of programmable memory (EPROM or OTPROM).

3.3.1 Memory Types

RAM means that any word in the memory may be accessed without having to go through all the other words to get to it. RAM is a volatile form of memory in that all the memory content is lost when the power is removed from the chip. RAM contents may be retained by keeping at least 2 volts on V_{DD}. Power requirements in this standby mode are very small.

ROM is very similar to RAM except, unlike RAM, it is not possible to change the contents of ROM after it is manufactured. This type memory is useful only for storage of information or programs.

The special bootstrap mode allows programs to be downloaded through the on-chip serial communications interface (SCI) into internal RAM to be executed. The bootloaded program is used for a variety of tasks such as loading calibration values into internal EPROM or performing diagnostics on a finished module.

The MC68HC705C8 on-chip ROM is called the bootloader ROM. This ROM controls the loading process of the special bootstrap mode.

Erasable programmable ROM (EPROM) is nonvolatile memory that can be programmed in the field by the user. Nonvolatile memories retain their contents even when no power is applied. Once it has been programmed, the EPROM cannot be written into, but it can be read from as many times as necessary. However, EPROM can be erased by ultraviolet light and reprogrammed.

OTPROM is the same as EPROM except it can be programmed only once and cannot be erased.

3.3.2 Memory Map

The MC68HC705C8 MCU contains four selectable memory configurations as shown in Figure 3-5. The memory configurations are accessed via the option

■ 6367248 0149795 615 ■

Register map ($00–$1F):

Register	Address
PORT A DATA REGISTER	$00
PORT B DATA REGISTER	$01
PORT C DATA REGISTER	$02
PORT D FIXED INPUT REGISTER	$03
PORT A DATA DIRECTION REG.	$04
PORT B DATA DIRECTION REG.	$05
PORT C DATA DIRECTION REG.	$06
UNUSED	$07
UNUSED	$08
UNUSED	$09
SPI CONTROL REGISTER	$0A
SPI STATUS REGISTER	$0B
SPI DATA I/O REGISTER	$0C
SCI BAUD RATE REGISTER	$0D
SCI CONTROL REGISTER 1	$0E
SCI CONTROL REGISTER 2	$0F
SCI STATUS REGISTER	$10
SCI DATA REGISTER	$11
TIMER CONTROL REGISTER	$12
TIMER STATUS REGISTER	$13
INPUT CAPTURE REG. (HIGH)	$14
INPUT CAPTURE REG. (LOW)	$15
OUTPUT COMPARE REG. (HIGH)	$16
OUTPUT COMPARE REG. (LOW)	$17
TIMER COUNT REGISTER (HIGH)	$18
TIMER COUNT REGISTER (LOW)	$19
ALT. COUNT REGISTER (HIGH)	$1A
ALT. COUNT REGISTER (LOW)	$1B
EPROM PROGRAM REGISTER	$1C
COP RESET REGISTER	$1D
COP CONTROL REGISTER	$1E
UNUSED	$1F

Vector map ($1FF4–$1FFF):

Vector	Address
SPI VECTOR (HIGH)	$1FF4
SPI VECTOR (LOW)	$1FF5
SCI VECTOR (HIGH)	$1FF6
SCI VECTOR (LOW)	$1FF7
TIMER VECTOR (HIGH)	$1FF8
TIMER VECTOR (LOW)	$1FF9
IRQ VECTOR (HIGH)	$1FFA
IRQ VECTOR (LOW)	$1FFB
SWI VECTOR (HIGH)	$1FFC
SWI VECTOR (LOW)	$1FFD
RESET VECTOR (HIGH BYTE)	$1FFE
RESET VECTOR (LOW BYTE)	$1FFF

* Refer to 3.10.4 OPTION REGISTER for an explanation of software-selectable memory configurations.

Figure 3-5. MC68HC705C8 Memory Map

6367248 0149796 551

register ($1FDF) RAM0 and RAM1 bits. During reset, the RAM0 and RAM1
control bits are forced to 0. RAM0 and RAM1 bit states determine the amount
of RAM and PROM, which can be selected as follows:

RAM0	RAM1	RAM Bytes	PROM Bytes
0	0	176	7744
1	0	208	7696
0	1	272	7648
1	1	304	7600

3.4 CENTRAL PROCESSOR UNIT

The MC68HC705C8 CPU is responsible for executing all software instructions
in their programmed sequence for a specific application.
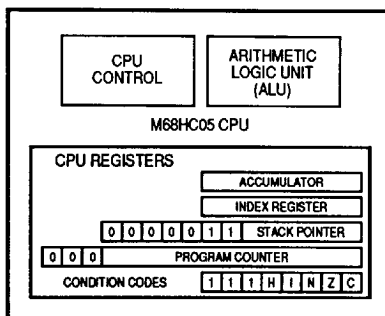
The CPU block diagram is shown in Figure 3-6.

Figure 3-6. M68HC05 CPU Block Diagram

3.4.1 Registers

The CPU contains five registers as shown in Figure 3-7. Registers in the CPU
are memories inside the microprocessor (not part of the memory map).

Accumulator (A)
The accumulator is an 8-bit general-purpose register used to hold oper-
ands, results of the arithmetic calculations, and data manipulations. It is
also directly accessible to the CPU for nonarithmetic operations. The ac-
cumulator is used during the execution of a program when the contents
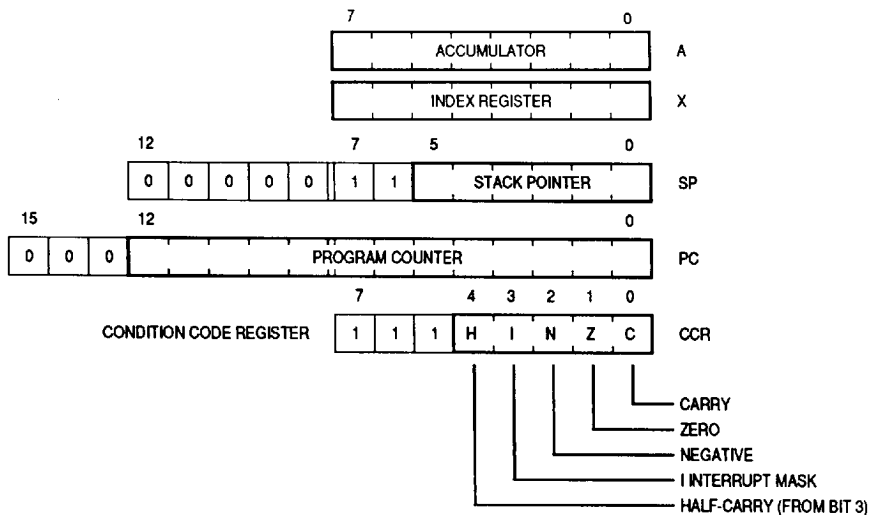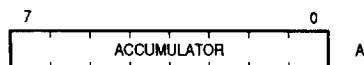
■ ьзь7248 0149797 498 ■

Figure 3-7. Programming Model

of some memory location are loaded into the accumulator. Also, the store instruction causes the contents of the accumulator to be stored at some prescribed memory location.

Index Register (X)

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

In indexed instructions, the X register provides an 8-bit value that is added to an instruction-provided value to create an effective address. The instruction-provided value can be 0, 1, or 2 bytes long.

■ 6367248 0149798 324 ■
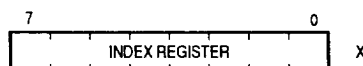
Condition Code Register (CCR)

The condition code register contains five status indicators that reflect the results of arithmetic and other operations of the CPU. The five flags are half-carry (H), negative (N), zero (Z), overflow (V), and carry/borrow (C).
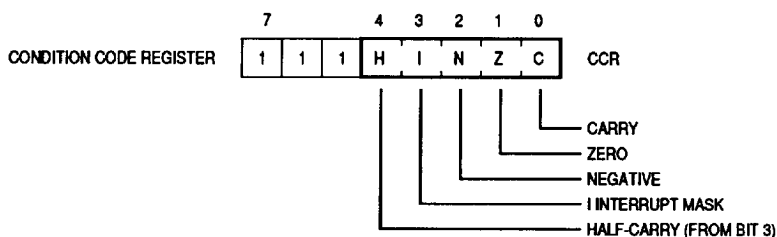
Half-Carry Bit (H) — The half-carry flag is used for binary-coded decimal (BCD) arithmetic operations and is affected by the ADD or ADC addition instructions. The H bit is set to a one when a carry occurs between bits 3 and 4.

Interrupt Mask Bit (I) — The interrupt mask bit disables all maskable interrupt sources when the I bit is set. Clearing this bit enables the interrupts. When any interrupt occurs, the I bit is automatically set after the registers are stacked but before the interrupt vector is fetched.

If an external interrupt occurs while the I bit is set, the interrupt is latched and processed after the I bit is cleared; therefore, no interrupts from the IRQ pin are lost because of the I bit being set.

After an interrupt has been serviced, a return from interrupt (RTI) instruction causes the registers to be restored to their previous values. Normally, the I bit would be zero after an RTI was executed. After any reset, I is set and can only be cleared by a software instruction.

Negative (N) — The N bit is set to one when the result of the last arithmetic, logical, or data manipulation is negative (bit 7 of the MSB in the result is a logic one).

The N bit has other uses. By assigning an often-tested flag bit to the MSB of a register or memory location, you can test this bit simply by loading the accumulator with the contents of that location.

Zero (Z) — The Z bit is set to one when the result of the last arithmetic, logical, or data manipulation is zero.

Carry/Borrow (C) — The C bit is used to indicate whether or not there was a carry from an addition or a borrow as a result of a subtraction. Shift and rotate instructions operate with and through the carry bit to facilitate multiple word shift operations. This bit is also affected during bit test and branch instructions.

The following illustration is an example of the way condition code bits are affected by arithmetic operations.

Assume Initial Values in Accumulator and Condition Codes:

ACCUMULATOR									CONDITION CODES					
7							0				H	I	N	Z	C
Before	1	1	1	1	1	1	1	1	($FF)	1	1	1	0	1	1	0	0
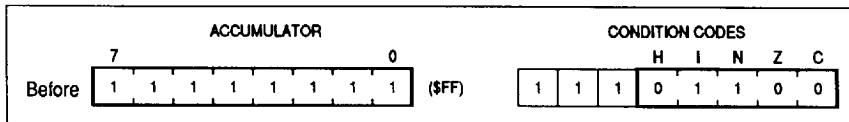
Execute the Following Instruction:

```
---- AB 02              ADD    #2    Add 2 to Accumulator
```

ACCUMULATOR									CONDITION CODES					
7							0				H	I	N	Z	C
After	0	0	0	0	0	0	0	1	($01)	1	1	1	1	1	0	0	1
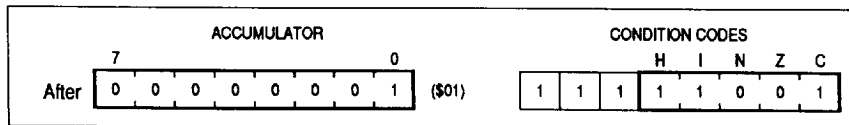
Condition Codes and Accumulator Reflect the Results of the Add Instruction:

H- Set because there was a carry from bit 3 to bit 4 of the accumulator.
I - No change.
N- Clear because result is not negative (bit 7 of accumulator is 0).
Z- Clear because result is not zero.
C- Set because there was a carry out of bit 7 of the accumulator.

The H bit is not useful after this operation because the accumulator was not a valid BCD value before the operation.

6367248 0149800 802

Program Counter (PC)

The program counter is a 13-bit register that contains the address of the next instruction or instruction operand to be fetched by the processor.

```
15        12                                              0
┌───┬───┬───┬─────────────────────────────────────────┐
│ 0 │ 0 │ 0 │            PROGRAM COUNTER              │  PC
└───┴───┴───┴─────────────────────────────────────────┘
```
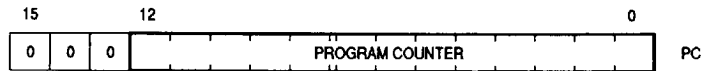
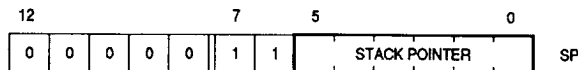Normally, the program counter advances one memory location at a time as instructions and instruction operands are fetched.

Jump, branch, and interrupt operations cause the program counter to be loaded with a memory address other than that of the next sequential location.

Stack Pointer (SP)

The stack pointer is a 13-bit register that contains the address of the next free location on the stack. During an MCU reset or the reset stack pointer (RSP) instruction, the stack pointer is set to location $00FF. The stack pointer is then decremented as data is pushed onto the stack and incremented as data is pulled from the stack.

```
12                      7     5                 0
┌───┬───┬───┬───┬───┬───┬───┬─────────────────┐
│ 0 │ 0 │ 0 │ 0 │ 0 │ 1 │ 1 │  STACK POINTER  │  SP
└───┴───┴───┴───┴───┴───┴───┴─────────────────┘
```

When accessing memory, the seven MSBs of the SP are permanently set to 0000011. These seven bits are appended to the six LSB bits to produce an address within the range of $00FF to $00CO. Subroutines and interrupts may use up to 64 (decimal) locations. If 64 locations are exceeded, the stack pointer wraps around and loses the previously stored information. A subroutine call occupies two locations on the stack; an interrupt uses five locations.

3.4.2 Arithmetic/Logic Unit (ALU)

The arithmetic/logic unit (ALU) is used to perform the arithmetic and logical operations defined by the instruction set.

The various binary arithmetic operations circuits decode the instruction in the instruction register and set up the ALU for the desired function. Most binary arithmetic is based on the addition algorithm, and subtraction is carried out as negative addition. Multiplication is not performed as a discrete instruction but as a chain of addition and shift operations within the ALU under control of CPU control logic. The multiply instruction (MUL) requires 11 internal processor cycles to complete this chain of operations.

3.4.3 CPU Control

The CPU control circuitry sequences the logic elements of the ALU to carry out the required operations.

3.4.4 Resets

Reset is used to force the MCU system to a known starting address. Peripheral systems and many control and status bits are also forced to a known state as a result of reset.

The following four conditions can cause reset in the MC68HC705C8 MCU:
1) External, active-low input signal on the $\overline{\text{RESET}}$ pin.
2) Internal power-on reset (POR) condition.
3) Internal computer operating properly (COP) watchdog system reset condition.
4) Internal clock monitor reset condition.

3.4.4.1 POWER-ON RESET.

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 voltage. There is no provision for a power-down reset.

The power-on circuitry provides for a 4064 cycle delay from the time that the oscillator becomes active. If the external $\overline{\text{RESET}}$ pin is low at the end of the 4064 delay timeout, the processor remains in the reset condition until $\overline{\text{RESET}}$ goes high.

■ ЬЗЬ7248 0149802 Ь85 ■

The following internal actions occur as the result of any MCU reset:

1) All data direction registers are cleared to zero (input).

2) Stack pointer configured to $00FF.

3) I bit in the condition code register to logic one.

4) External interrupt latch cleared.

5) SCI disabled (serial control bits TE = 0 and RE = 0). Other SCI bits cleared by reset include: TIE, TCIE, RIE, ILIE, RWU, SBK, RDRF, IDLE, OR, NF, and FE.

6) Serial status bits TDRE and TC set.

7) SCI prescaler and rate control bits SCP0, SCP1 cleared.

8) SPI disable (serial output enable control bit SPE = 0). Other SPI bits cleared by reset include: SPIE, MSTR, SPIF, WCOL, and MODF.

9) All serial interrupt enable bits cleared (SPIE, TIE, and TCIE).

10) SPI system configured as slave (MSTR = 0).

11) Timer prescaler reset to zero state.
    Timer counter configured to $FFFC.
    Timer output compare (TCMP) bit reset to zero.
    All timer interrupt enable bits cleared (ICIE, OCIE, and TOIE) to disable timer interrupts.
    The OLVL timer bit is also cleared by reset.

12) STOP latch cleared.

13) WAIT latch cleared.

14) Internal address bus forced to restart vector (on exit from reset, upper byte of program counter is loaded from $1FFE, and lower byte of program counter is loaded from $1FFF).

3.4.4.2 COMPUTER OPERATING PROPERLY (COP) WATCHDOG TIMER RESET. The COP watchdog timer system is intended to detect software errors. When the COP is being used, software is responsible for keeping a free-running watchdog timer from timing out. If the watchdog timer times out, it is an indication that software is no longer being executed in the intended sequence; thus, a system reset is initiated.

■ 6367248 0149803 511 ■

Since the COP timer relies on the internal bus clock in order to detect a software failure, a clock monitor is also included to guard against a failure of the clock. When the COP timer is enabled, the clock monitor should also be enabled since the COP timer cannot detect failures of the internal bus clock.

The COP control register ($1E), as shown below, is used to control the COP watchdog timer and clock monitor functions.

[1] - Cleared on external or POR reset, set on COP or clock monitor fail resets.
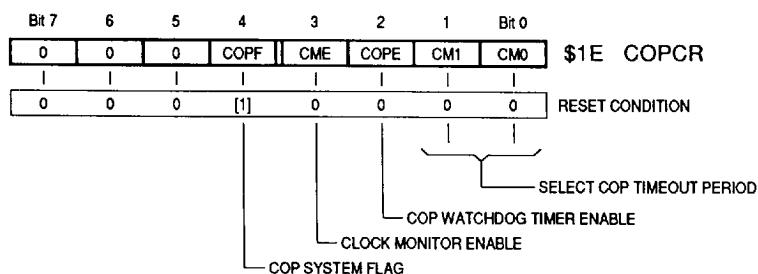
COPF — Computer Operating Properly Flag
   1 = COP or clock monitor reset has occurred
   0 = No COP or clock monitor reset has occurred
        Reading the COP control register clears COPF.

CME — Clock Monitor Enable
   1 = Clock monitor enabled
   0 = Clock monitor disabled
        CME is readable and writable at any time.

COPE — Computer Operating Properly Enable
   1 = COP timeout enabled
   0 = COP timeout disabled

CM1, CM0 — Computer Operating Properly Mode
        These two bits are used to select the COP watchdog timeout period
   (see Table 3-1).

The actual timeout period is dependent on the system bus clock frequency, but, for reference purposes, Table 3-1 shows the relationship between the CM1 and CM0 select bits and the COP timeout period for various system

Table 3-1. COP Timeout Period versus CM1 and CM0

CM1	CM0	E/2^{15} Div. By	XTAL = 4.0 MHz E = 2.0 MHz Timeout	XTAL = 3.5796 E = 1.7897 MHZ Timeout	XTAL = 2.0 MHz E = 1.0 MHz Timeout	XTAL = 1.0 MHz E = 0.5 MHz Timeout
0	0	1	16.38 ms	18.31 ms	32.77 ms	65.54 ms
0	1	4	65.54 ms	73.24 ms	131.07 ms	262.14 ms
1	0	16	262.14 ms	292.95 ms	524.29 ms	1.048 s
1	1	64	1.048 s	1.172 s	2.097 s	4.194 s

clock frequencies ("E" stands for the system bus clock). The default reset condition of the COP mode bits (CM1 and CM0) is cleared, which corresponds to the shortest timeout period.

The COP reset register ($1D) is used to keep the COP watchdog timer from timing out.
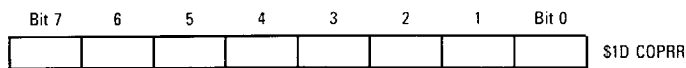
The sequence required to reset the COP watchdog timer is:
1) Write $55 to the COP reset register at location $1D.
2) Write $AA to the same address location.

Both write operations must occur in the correct order prior to timeout, but any number of instructions may be executed between the two write operations. The elapsed time between adjacent software reset sequences must never be greater than the COP timeout period.

Upon detection of a timeout condition, the COP watchdog timer (if enabled by COPE = 1) will cause a system reset to be generated. This reset is issued to the external system via the bidirectional $\overline{\text{RESET}}$ pin for four bus cycles.

3.4.4.3 CLOCK MONITOR RESET. When a clock failure is detected by the clock monitor (and CME = 1), a system reset will be generated.

When CME is set, the clock monitor detects the absence of the internal bus clock for more than a certain period of time. When CME is cleared, the clock monitor is disabled. The timeout period is dependent on processing parameters and will be between 5 and 100 µs. Thus, a bus clock rate of 200 kHz or more will never cause a clock monitor failure, and a bus clock rate of 10 kHz or less will definitely cause a clock monitor reset.

■ 6367248 0149805 394 ■

A clock monitor reset is issued to the external system via the bidirectional RESET pin for four bus cycles. The clock monitor does not have a separate reset vector.

Special considerations are needed when using the STOP instruction with the clock monitor. Since the STOP instruction causes the clocks to be halted, the clock monitor will generate a reset sequence (if enabled by CME = 1) at the time the STOP instruction is entered.

3.4.5 Addressing Modes

The power of any computer lies in its ability to access memory. The addressing modes of the CPU provide that capability. The addressing modes define the manner in which an instruction is to obtain the data required for its execution. Because of different addressing modes, an instruction may access the operand in one of up to six different ways. In this manner, the addressing modes expand the basic 62 M68HC05 Family instructions into 210 distinct opcodes.

The M68HC05 addressing modes that are used to reference memory are inherent, immediate, extended, direct, indexed (no offset, 8-bit offset, and 16-bit offset), and relative. One- and two-byte direct addressing instructions access all data bytes in most applications. Extended addressing uses three-byte instructions to reach data anywhere in memory space. The various addressing modes make it possible to locate data tables, code conversion tables, and scaling tables anywhere in the memory space. Short indexed accesses are single-byte instructions; whereas, the longest instructions (three bytes) permit accessing tables anywhere in memory.

A general description and examples of the various modes of addressing are provided in the following paragraphs. The term effective address (EA) is used to indicate the memory address where the argument for an instruction is fetched or stored. More details on addressing modes and a description of each instruction is available in Appendix A.

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

1. A blank or no symbol indicates a decimal number.

2. A $ immediately preceding a number indicates it is a hexadecimal number; e.g., $24 is 24 in hexadecimal or the equivalent of 36 in decimal.

■ 6367248 0149806 220 ■

3. A # indicates immediate operand and the number is found in the location following the opcode. A variety of symbols and expressions can be used following the character # sign. Since not all assemblers use the same syntax rules and special characters, refer to the documentation for the particular assembler that will be used.

Prefix	Definition
None	Decimal
$	Hexadecimal
(α	Octal
%	Binary
'	Single ASCII Character

For each addressing mode, an example instruction is explained in detail. These explanations describe what happens in the CPU during each processor clock cycle of the instruction. Numbers in square brackets [ ] refer to a specific CPU clock cycle.

3.4.5.1 INHERENT ADDRESSING MODE. In inherent addressing mode, all information required for the operation is already inherently known to the CPU, and no external operand from memory or from the program is needed. The operands (if any) are only the index register and accumulator. These are always one byte instructions.

Example Program Listing:
```
0200 4c                    INCA              Increment accumulator
```

Execution Sequence:
    $0200 $4C   [1], [2], [3]

Explanation:
    [1]   CPU reads opcode $4C — increment accumulator
    [2], [3] CPU reads accumulator value, adds one to it, stores the new value in the accumulator, and adjusts condition code flag bits as necessary.

■ 6367248 0149807 167 ■

The following is a list of all M68HC05 instructions that can use the inherent addressing mode.

Instruction	Mnemonic
Arithmetic Shift Left	ASLA, ASLX
Arithmetic Shift Right	ASRA, ASRX
Clear Carry Bit	CLC
Clear Interrupt Mask Bit	CLI
Clear	CLRA, CLRX
Complement	COMA, COMX
Decrement	DECA, DECX
Increment	INCA, INCX
Logical Shift Left	LSLA, LSLX
Logical Shift Right	LSRA, LSRX
Multiply	MUL
Negate	NEGA, NEGX
No Operation	NOP
Rotate Left thru Carry	ROLA, ROLX
Rotate Right thru Carry	RORA, RORX
Reset Stack Pointer	RSP
Return from Interrupt	RTI
Return from Subroutine	RTS
Set Carry Bit	SEC
Set Interrupt Mask Bit	SEI
Enable IRQ, Stop Oscillator	STOP
Software Interrupt	SWI
Transfer Accumulator to Index Register	TAX
Test for Negative or Zero	TSTA, TSTX
Transfer Index Register to Accumulator	TXA
Enable Interrupt, Halt Processor	WAIT

3

■ 6367248 0149808 0T3 ■

3.4.5.2 IMMEDIATE ADDRESSING MODE.
In the immediate addressing mode, the operand is contained in the byte immediately 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.

Example Program Listing:

```
0200 a6 02   LDA  #$02  Load accumulator w/ immediate value
```

Execution Sequence:

$0200	$A6	[1]
$0201	$02	[2]

Explanation:
 [1] CPU reads opcode $A6 — load accumulator with the value immediately following the opcode.
 [2] CPU then reads the immediate data $02 from location $0201 and loads $02 into the accumulator.

The following is a list of all M68HC05 instructions that can use the immediate addressing mode.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Subtract	SUB

■ 6367248 0149809 T3T ■

3.4.5.3 EXTENDED ADDRESSING MODE. In the extended addressing mode, the address of the operand is contained in the two bytes following the opcode. Extended addressing references any location in the MCU memory space including I/O, RAM, ROM, and EPROM. Extended addressing mode instructions are three bytes, one for the opcode and two for the address of the operand.

Example Program Listing:

```
0200 c6 06 e5  LDA $06E5  Load accumulator from extended addr
```

Execution Sequence:

$0200 $C6   [1]
$0201 $06   [2]
$0202 $E5   [3] and [4]

Explanation:

[1] CPU reads opcode $C6 — load accumulator using extended addressing mode.

[2] CPU then reads $06 from location $0201. This $06 is interpreted as the high-order half of an address.

[3] CPU then reads $E5 from location $0202. This $E5 is interpreted as the low-order half of an address.

[4] CPU internally appends $06 to the $E5 read to form the complete address ($06E5). The CPU then reads whatever value is contained in the location $06E5 into the accumulator.

The following is a list of all M68HC05 instructions that can use the extended addressing mode.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Acumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB

3

■ 6367248 0149810 751 ■

3.4.5.4 DIRECT ADDRESSING MODE. The direct addressing mode is similar to the extended addressing mode except the upper byte of the operand address is assumed to be $00. Thus, only the lower byte of the operand address needs to be included in the instruction. Direct addressing allows you to efficiently address the lowest 256 bytes in memory. This area of memory is called the direct page and includes on-chip RAM and I/O registers. Direct addressing is efficient in both memory and time. Direct addressing mode instructions are usually two bytes, one for the opcode and one for the low-order byte of the operand address.

Example Program Listing:

```
0200 b6 50   LDA    $50    Load accumulator from direct address
```

Execution Sequence:
$0200 $B6   [1]
$0201 $50   [2] and [3]

Explanation:
[1] CPU reads opcode $B6 — load accumulator using direct addressing mode.
[2] CPU then reads $50 from location $0201. This $50 is interpreted as the low-order half of an address. In direct addressing mode, the high-order half of the address is assumed to be $00.
[3] CPU internally appends $00 to the $50 read in the second cycle to form the complete address ($0050). The CPU then reads whatever value is contained in the location $0050 into the accumulator.

■ 6367248 0149811 698 ■

The following is a list of all M68HC05 instructions that can use the direct addressing mode.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Clear Bit in Memory	BCLR
Bit Test Memory with Accumulator	BIT
Branch if Bit n is Clear	BRCLR
Branch if Bit n is Set	BRSET
Set Bit in Memory	BSET
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR
Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

3

6367248 0149812 524

3.4.5.5 INDEXED ADDRESSING MODE. In the indexed addressing mode, the effective address 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 MCU: no offset, 8-bit offset, and 16-bit offset. A good assembler should use the indexed addressing mode that requires the least number of bytes to express the offset.

3.4.5.5.1 Indexed, No Offset. In the indexed, no-offset addressing mode, the effective address of the instruction is contained in the 8-bit index register. Thus, this addressing mode can access the first 256 memory locations. These instructions are only one byte.

Example Program Listing:
```
0200 f6   LDA   ,X        Load accumulator from location
                          pointed to by index reg (no offset)
```

Execution Sequence:
$0200 $F6   [1],   [2],   [3]

Explanation:
[1] CPU reads opcode $F6 — load accumulator using indexed, no offset, addressing mode.
[2] CPU forms a complete address by adding $0000 to the contents of the index register.
[3] CPU then reads the contents of the addressed location into the accumulator.

The following is a list of all M68HC05 instructions that can use the indexed, no-offset addressing mode.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Bit Test Memory with Accumulator	BIT
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR
Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

3

■ 6367248 0149814 3T7 ■

3.4.5.5.2 Indexed, 8-Bit Offset. In the indexed, 8-bit offset addressing mode, the effective address is obtained by adding the contents of the byte following the opcode to the contents of the index register. This mode of addressing is useful for selecting the kth element in a '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 which the instruction may access). 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. The content of the index register (X) is not changed. The offset byte supplied in the instruction is an unsigned 8-bit integer.

Example Program Listing:

```
0200 e6 05  LDA    $5,X      Load accumulator from location
                             pointed to by index reg (X) + $05
```

Execution Sequence:
```
$0200   $E6  [1]
$0201   $05  [2],  [3],  [4]
```

Explanation:
[1] CPU reads opcode $E6 — load accumulator using indexed, 8-bit offset addressing mode.
[2] CPU then reads $05 from location $0201. This $05 is interpreted as the low-order half of a base address. The high-order half of the base address is assumed to be $00.
[3] CPU will add the value in the index register to the base address $0005. The results of this addition is the address that the CPU will use in the load accumulator operation.
[4] The CPU will then read the value from this address and load this value into the accumulator.

The following is a list of all M68HC05 instructions that can use the indexed, 8-bit offset addressing mode.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Bit Test Memory with Accumulator	BIT
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR
Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

3

■ ᏏᎦᏏ7ᕇᏏᎦ ᎤᏏᏏᏩᏏᏏᏏ Ꮟ7T ■

3.4.5.5.3 Indexed, 16-Bit Offset. In the indexed, 16-bit offset addressing mode, the effective address is the sum of the contents of the 8-bit index register and the two bytes following the opcode. The content of the index register is not changed. These instructions are three bytes, one for the opcode and two for a 16-bit offset.

Example Program Listing:

```
0200 d6 07 00 LDA $0700,X Load accumulator from location
                          pointed to by index reg (X) + $0700
```

Execution Sequence:
    $0200 $D6   [1]
    $0201 $07   [2]
    $0202 $00   [3], [4], [5]

Explanation:
    [1] CPU reads opcode $D6 — load accumulator using indexed, 16-bit offset addressing mode.
    [2] CPU then reads $07 from location $0201. This $07 is interpreted as the high-order half of a base address.
    [3] CPU then reads $00 from location $0202. This $00 is interpreted as the low-order half of a base address.
    [4] CPU will add the value in the index register to the base address $0700. The results of this addition is the address that the CPU will use in the load accumulator operation.
    [5] The CPU will then read the value from this address and load this value into the accumulator.

■ 6367248 0149817 006 ■

The following is a list of all M68HC05 instructions that can use the indexed, 16-bit offset addressing mode.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register In Memory	STX
Subtract	SUB

3

■ 6367248 0149818 T42 ■

3.4.5.6 RELATIVE ADDRESSING MODE. The relative addressing mode is used only for branch instructions. Branch instructions, other than the branching versions of bit-manipulation instructions, generate two machine-code bytes: one for the opcode and one for the relative offset. Because it is desirable to branch in either direction, the offset byte is a signed twos-complement offset with a range of −127 to +128 bytes (with respect to the address of the instruction immediately following the branch instruction). If the branch condition is true, the contents of the 8-bit signed byte following the opcode (offset) are added to the contents of the program counter to form the effective branch address; otherwise, control proceeds to the instruction immediately following the branch instruction.

A programmer specifies the destination of a branch as an absolute address (or label which refers to an absolute address). The Motorola assembler calculates the 8-bit signed relative offset, which is placed after the branch opcode in memory.

Example Program Listing:
```
   0200 27 rr          BEQ    DEST      Branch to DEST if Z=1
                                        (branch if equal or zero)
```
Execution Sequence:
```
  $0200   $27   [1]
  $0201   $rr   [2],  [3]
```

Explanation:
   [1] CPU reads opcode $27 — branch if Z=1, (relative addressing mode).
   [2] CPU reads the offset, $rr.
   [3] CPU internally tests the state of the Z bit and causes a branch if Z is set.

■ 6367248 0149819 989 ■

The following is a list of all M68HC05 instructions that can use the relative addressing mode.

Instruction	Mnemonic
Branch if Carry Clear	BCC
Branch is Carry Set	BCS
Branch if Equal	BEQ
Branch if Half-Carry Clear	BHCC
Branch if Half-Carry Set	BHCS
Branch if Higher	BHI
Branch if Higher or Same	BHS
Branch if Interrupt Line is High	BIH
Branch if Interrupt Line is Low	BIL
Branch if Lower	BLO
Branch if Lower or Same	BLS
Branch if Interrupt Mask is Clear	BMC
Branch if Minus	BMI
Branch if Interrupt Mask Bit is Set	BMS
Branch if Not Equal	BNE
Branch if Plus	BPL
Branch Always	BRA
Branch if Bit n is Clear	BRCLR
Branch if Bit n is Set	BRSET
Branch Never	BRN
Branch to Subroutine	BSR

3.4.5.7 BIT TEST AND BRANCH INSTRUCTIONS. These instructions use direct addressing mode to specify the location being tested and relative addressing to specify the branch destination. This applications guide treats these instructions as direct addressing mode instructions. Some older Motorola documents call the addressing mode of these instructions BTB for bit test and branch.

3.4.5.8 INSTRUCTIONS ORGANIZED BY TYPE. Tables 3-2 through 3-5 show the MC68HC05 instruction set displayed by instruction type.

■ 6367248 0149820 6T0 ■

Table 3-2. Register/Memory Instructions

Function	Mnem.	Immediate Op-code	# Bytes	# Cycles	Direct Op-code	# Bytes	# Cycles	Extended Op-code	# Bytes	# Cycles	Indexed (No Offset) Op-code	# Bytes	# Cycles	Indexed (8-Bit Offset) Op-code	# Bytes	# Cycles	Indexed (16-Bit Offset) 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

■ 6367248 0149821 537 ■

Table 3-3. Read/Modify-Write Instructions

Function	Mnem.	Inherent (A) Op-code	# Bytes	# Cycles	Inherent (X) Op-code	# Bytes	# Cycles	Direct Op-code	# Bytes	# Cycles	Indexed (No Offset) Op-code	# Bytes	# Cycles	Indexed (8-Bit Offset) 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	RCL	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	—	—	—	—	—	—	—	—	—	—	—	—

3

■ 6367248 0149822 473 ■

Table 3-4. Branch Instructions

Function	Mnemonic	Relative Addressing Mode		
		Opcode	# 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 (Same as BCC)	BHS	24	2	3
Branch IFF Carry Set	BCS	25	2	3
Branch IFF Lower (Same as BCS)	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

3

■ 6367248 0149823 30T ■

Table 3-5. Control Instructions

Function	Mnemonic	Inherent		
		Opcode	# 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

3

■ 6367248 0149824 246 ■

3.4.6 Instruction Set Summary

Computers use an operation code or opcode to give instructions to the CPU. The instruction set for a specific CPU is the set of all opcodes that the CPU knows how to execute. The CPU in the MC68HC705C8 MCU can understand 62 basic instructions, some of which have several variations that require separate opcodes. The M68HC05 instruction set includes 210 unique instruction opcodes.

The following table is an alphabetical listing of the M68HC05 instructions available to the user. In listing all the factors necessary to program, the table uses the following symbols:

Condition Code Symbols

H	— Half Carry (Bit 4)	♠	— Test and Set if True,
I	— Interrupt Mask (Bit 3)		(cleared otherwise)
N	— Negate (Sign Bit 2)	—	— Not Affected
Z	— Zero (Bit 1)	?	— Load CC from Stack
C	— Carry/Borrow (Bit 0)	0	— Cleared
		1	— Set

Boolean Operators
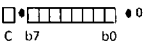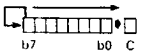
( )	— Contents of (i.e., (M)	+	— (inclusive) OR
	means the contents	⊕	— Exclusive OR
	of memory location	—	— NOT
	M)	–	— Negation
♦	— is loaded with, 'gets'		(twos complement)
●	— AND	×	— Multiplication

MPU Registers

A	— Accumulator	PC	— Program Counter
ACCA	— Accumulator	PCH	— PC High Byte
CC	— Condition Code Reg.	PCL	— PC Low Byte
X	— Index Register	SP	— Stack Pointer
M	— Any memory location	REL	— Relative Address
	(one byte)		

Addressing Modes	(Abbreviation)	Operands	
Inherent	INH	none	
Immediate	IMM	ii	
Direct (for bit	DIR	dd	
test instructions)		dd	rr
Extended	EXT	hh	ll
Indexed 0 Offset	IX	none	
Indexed 1-Byte	IX1	ff	
Indexed 2-Byte	IX2	ee	ff
Relative	REL	rr	

■ 6367248 0149825 182 ■

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (hexadecimal) Opcode	Operand	Bytes	Cycles	H	I	N	Z	C
ADC (opr)	Add with Carry	ACCA ⬦ ACCA + M + C	IMM	A9	ii	2	2	‡	—	‡	‡	‡
			DIR	B9	dd	2	3					
			EXT	C9	hh ll	3	4					
			IX2	D9	ee ff	3	5					
			IX1	E9	ff	2	4					
			IX	F9		1	3					
ADD (opr)	Add	ACCA ⬦ ACCA + M	IMM	AB	ii	2	2	‡	—	‡	‡	‡
			DIR	BB	dd	2	3					
			EXT	CB	hh ll	3	4					
			IX2	DB	ee ff	3	5					
			IX1	EB	ff	2	4					
			IX	FB		1	3					
AND (opr)	Logical AND	ACCA ⬦ ACCA • M	IMM	A4	ii	2	2	—	—	‡	‡	—
			DIR	B4	dd	2	3					
			EXT	C4	hh ll	3	4					
			IX2	D4	ee ff	3	5					
			IX1	E4	ff	2	4					
			IX	F4		1	3					
ASL (opr) ASLA ASLX ASL (opr) ASL (opr)	Arithmetic Shift Left		DIR	38	dd	2	5	—	—	‡	‡	‡
			INH(A)	48		1	3					
			INH(X)	58		1	3					
			IX1	68	ff	2	6					
			IX	78		1	5					
ASR (opr) ASRA ASRX ASR (opr) ASR (opr)	Arithmetic Shift Right		DIR	37	dd	2	5	—	—	‡	‡	‡
			INH(A)	47		1	3					
			INH(X)	57		1	3					
			IX1	67	ff	2	6					
			IX	77		1	5					
BCC (rel)	Branch if Carry Clear	? C = 0	REL	24	rr	2	3	—	—	—	—	—
BCLR n, (opr)	Clear Bit n in Memory	Mn ⬦ 0	DIR(b0)	11	dd	2	5	—	—	—	—	—
			DIR(b1)	13	dd	2	5					
			DIR(b2)	15	dd	2	5					
			DIR(b3)	17	dd	2	5					
			DIR(b4)	19	dd	2	5					
			DIR(b5)	1B	dd	2	5					
			DIR(b6)	1D	dd	2	5					
			DIR(b7)	1F	dd	2	5					
BCS (rel)	Branch if Carry Set	? C = 1	REL	25	rr	2	3	—	—	—	—	—
BEQ (rel)	Branch if Equal	? Z = 1	REL	27	rr	2	3	—	—	—	—	—
BHCC (rel)	Branch if Half Carry Clear	? H = 0	REL	28	rr	2	3	—	—	—	—	—
BHCS (rel)	Branch if Half Carry Set	? H = 1	REL	29	rr	2	3	—	—	—	—	—
BHI (rel)	Branch if Higher	? (C + Z) = 0	REL	22	rr	2	3	—	—	—	—	—
BHS (rel)	Branch if Higher or Same	? C = 0	REL	24	rr	2	3	—	—	—	—	—
BIH (rel)	Branch if IRQ Pin is High	? IRQ Pin = 1	REL	2F	rr	2	3	—	—	—	—	—
BIL (rel)	Branch if IRQ Pin is Low	? IRQ Pin = 0	REL	2E	rr	2	3	—	—	—	—	—
BIT (rel)	Bit Test Memory with A	ACCA • M	IMM	A5	ii	2	2	—	—	‡	‡	—
			DIR	B5	dd	2	3					
			EXT	C5	hh ll	3	4					
			IX2	D5	ee ff	3	5					
			IX1	E5	ff	2	4					
			IX	F5		1	3					
BLO (rel)	Branch if Lower	? C = 1	REL	25	rr	2	3	—	—	—	—	—
BLS (rel)	Branch if Lower or Same	? (C + X) = 1	REL	23	rr	2	3	—	—	—	—	—
BMC (rel)	Branch if I Bit is Clear	? I = 0	REL	2C	rr	2	3	—	—	—	—	—
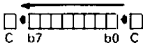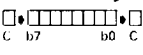
3

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (hexadecimal) Opcode	Operand	Bytes	Cycles	Condition Code H	I	N	Z	C
BMI (rel)	Branch if Minus	? N – 1	REL	2B	rr	2	3	—	—	—	—	—
BMS (rel)	Branch if I Bit is Set	? I – 1	REL	2D	rr	2	3	—	—	—	—	—
BNE (rel)	Branch if Not Equal	? Z – 0	REL	26	rr	2	3	—	—	—	—	—
BPL (rel)	Branch if Plus	? N – 0	REL	2A	rr	2	3	—	—	—	—	—
BRA (rel)	Branch Always	? 1 – 1	REL	20	rr	2	3	—	—	—	—	—
BRCLR n, (opr) (rel)	Branch if Bit n of M – 0	? Bit n of M – 0	DIR(b0)	01	dd rr	3	5	—	—	—	—	↕
			DIR(b1)	03	dd rr	3	5					
			DIR(b2)	05	dd rr	3	5					
			DIR(b3)	07	dd rr	3	5					
			DIR(b4)	09	dd rr	3	5					
			DIR(b5)	0B	dd rr	3	5					
			DIR(b6)	0D	dd rr	3	5					
			DIR(b7)	0F	dd rr	3	5					
BRN (rel)	Branch Never	? 1 – 0	REL	21	rr	2	3	—	—	—	—	—
BRSET n, (opr) (rel)	Branch if Bit n of M – 1	? Bit n of M – 1	DIR(b0)	00	dd rr	3	5	—	—	—	—	↕
			DIR(b1)	02	dd rr	3	5					
			DIR(b2)	04	dd rr	3	5					
			DIR(b3)	06	dd rr	3	5					
			DIR(b4)	08	dd rr	3	5					
			DIR(b5)	0A	dd rr	3	5					
			DIR(b6)	0C	dd rr	3	5					
			DIR(b7)	0E	dd rr	3	5					
BSET n, (opr)	Set Bit n in Memory	Mn ← 1	DIR(b0)	10	dd	2	5	—	—	—	—	—
			DIR(b1)	12	dd	2	5					
			DIR(b2)	14	dd	2	5					
			DIR(b3)	16	dd	2	5					
			DIR(b4)	18	dd	2	5					
			DIR(b5)	1A	dd	2	5					
			DIR(b6)	1C	dd	2	5					
			DIR(b7)	1E	dd	2	5					
BSR (rel)	Branch to Subroutine	PC ← PC + 0002 (SP) ← PCL; SP ← SP – 0001 (SP) ← PCH; SP ← SP – 0001 PC ← PC + Rel	REL	AD	rr	2	6	—	—	—	—	—
CLC	Clear C Bit	C bit ← 0	INH	98		1	2	—	—	—	—	0
CLI	Clear I Bit	I bit ← 0	INH	9A		1	2	—	0	—	—	—
CLR (opr) CLRA CLRX CLR (opr) CLR (opr)	Clear	M ← 00 A ← 00 X ← 00 M ← 00 M ← 00	DIR INH(A) INH(X) IX1 IX	3F 4F 5F 6F 7F	dd ff	2 1 1 2 1	5 3 3 6 5	—	—	0	1	—
CMP (opr)	Compare A with Memory	ACCA – M	IMM DIR EXT IX2 IX1 IX	A1 B1 C1 D1 E1 F1	ii dd hh ll ee ff ff	2 2 3 3 2 1	2 3 4 5 4 3	—	—	↕	↕	↕
COM (opr) COMA COMX COM (opr) COM (opr)	1's Complement	M ← M̄ – $FF – M A ← Ā – $FF – A X ← X̄ – $FF – X M ← M̄ – $FF – M M ← M̄ – $FF – M	DIR INH(A) INH(X) IX1 IX	33 43 53 63 73	dd ff	2 1 1 2 1	5 3 3 6 5	—	—	↕	↕	1
CPX (opr)	Compare X with Memory	X – M	IMM DIR EXT IX2 IX1 IX	A3 B3 C3 D3 E3 F3	ii dd hh ll ee ff ff	2 2 3 3 2 1	2 3 4 5 4 3	—	—	↕	↕	↕

■ 6367248 0149827 T55 ■

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (hexadecimal) Opcode	Operand	Bytes	Cycles	H	I	N	Z	C
DEC (opr) DECA DECX DEC (opr) DEC (opr)	Decrement DEX (same as DECX)	M ⬦ M – 01 A ⬦ A – 01 X ⬦ X – 01 M ⬦ M – 01 M ⬦ M – 01	DIR INH(A) INH(X) IX1 IX	3A 4A 5A 6A 7A	dd ff 	2 1 1 2 1	5 3 3 6 5	—	—	⇕	⇕	—
EOR (opr)	Exclusive OR A with Memory	ACCA ⬦ ACCA ⊕ M	IMM DIR EXT IX2 IX1 IX	A8 B8 C8 D8 E8 F8	ii dd hh ll ee ff ff 	2 2 3 3 2 1	2 3 4 5 4 3	—	—	⇕	⇕	—
INC (opr) INCA INCX INC (opr) INC (opr)	Increment INX (same as INCX)	M ⬦ M + 01 A ⬦ A + 01 X ⬦ X + 01 M ⬦ M + 01 M ⬦ M + 01	DIR INH(A) INH(X) IX1 IX	3C 4C 5C 6C 7C	dd ff 	2 1 1 2 1	5 3 3 6 5	—	—	⇕	⇕	—
JMP (opr)	Jump	PC ⬦ effective address	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff 	2 3 3 2 1	2 3 4 3 2	—	—	—	—	—
JSR (opr)	Jump to Subroutine	PC ⬦ PC + n (n – 1, 2, or 3) (SP) ⬦ PCL; SP ⬦ SP – 0001 (SP) ⬦ PCH; SP ⬦ SP – 0001 PC ⬦ effective address	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff 	2 3 3 2 1	5 6 7 6 5	—	—	—	—	—
LDA (opr)	Load A from Memory	ACCA ⬦ M	IMM DIR EXT IX2 IX1 IX	A6 B6 C6 D6 E6 F6	ii dd hh ll ee ff ff 	2 2 3 3 2 1	2 3 4 5 4 3	—	—	⇕	⇕	—
LDX (opr)	Load X from Memory	X ⬦ M	IMM DIR EXT IX2 IX1 IX	AE BE CE DE EE FE	ii dd hh ll ee ff ff 	2 2 3 3 2 1	2 3 4 5 4 3	—	—	⇕	⇕	—
LSL (opr) LSLA LSLX LSL (opr) LSL (opr)	Logical Shift Left	(see diagram)	DIR INH(A) INH(X) IX1 IX	38 48 58 68 78	dd ff 	2 1 1 2 1	5 3 3 6 5	—	—	⇕	⇕	⇕
LSR (opr) LSRA LSRX LSR (opr) LSR (opr)	Logical Shift Right	(see diagram)	DIR INH(A) INH(X) IX1 IX	34 44 54 64 74	dd ff 	2 1 1 2 1	5 3 3 6 5	—	—	0	⇕	⇕
MUL	Unsigned Multiply	X:A ⬦ X • A	INH	42		1	11	0	—	—	—	0
NEG (opr) NEGA NEGX NEG (opr) NEG (opr	Negate (2's Complement)	M ⬦ – M (i.e. 00 – M) A ⬦ – A X ⬦ – X M ⬦ – M M ⬦ – M	DIR INH(A) INH(X) IX1 IX	30 40 50 60 70	dd ff 	2 1 1 2 1	5 3 3 6 5	—	—	⇕	⇕	⇕
NOP	No Operation		INH	9D		1	2	—	—	—	—	—
ORA (opr)	Inclusive OR	ACCA ⬦ ACCA + M	IMM DIR EXT IX2 IX1 IX	AA BA CA DA EA FA	ii dd hh ll ee ff ff 	2 2 3 3 2 1	2 3 4 5 4 3	—	—	⇕	⇕	—

3

■ 6367248 0149828 991 ■

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Opcode	Operand	Bytes	Cycles	H	I	N	Z	C
ROL (opr) ROLA ROLX ROL (opr) ROL (opr	Rotate Left through Carry		DIR INH(A) INH(X) IX1 IX	39 49 59 69 79	dd ff	2 1 1 2 1	5 3 3 6 5	—	—	⇕	⇕	⇕
ROR (opr) RORA RORX ROR (opr) ROR (opr)	Rotate Right through Carry		DIR INH(A) INH(X) IX1 IX	36 46 56 66 76	dd ff	2 1 1 2 1	5 3 3 6 5	—	—	⇕	⇕	⇕
RSP	Reset Stack Pointer	SP ⬥ $00FF	INH	9C		1	2	—	—	—	—	—
RTI	Return from Interrupt	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)	INH	80		1	9	(From Stack) ⇕	⇕	⇕	⇕	⇕
RTS	Return from Subroutine	SP ⬥ SP + 0001; PCH ⬥ (SP) SP ⬥ SP + 0001; PCL ⬥ (SP)	INH	81		1	6	—	—	—	—	—
SBC (opr)	Subtract with Carry	ACCA ⬥ ACCA - M - C	IMM DIR EXT IX2 IX1 IX	A2 B2 C2 D2 E2 F2	ii dd hh ll ee ff ff	2 2 3 3 2 1	2 3 4 5 4 3	—	—	⇕	⇕	⇕
SEC	Set C Bit	C bit ⬥ 1	INH	99		1	2	—	—	—	—	1
SEI	Set I Bit	I bit ⬥ 1	INH	9B		1	2	—	1	—	—	—
STA (opr)	Store A in Memory	M ⬥ ACCA	DIR EXT IX2 IX1 IX	B7 C7 D7 E7 F7	dd hh ll ee ff ff	2 3 3 2 1	4 5 6 5 4	—	—	⇕	⇕	—
STOP	Enable IRQ, Stop Oscillator		INH	8E		1	2	—	0	—	—	—
STX (opr)	Store X in Memory	M ⬥ X	DIR EXT IX2 IX1 IX	BF CF DF EF FF	dd hh ll ee ff ff	2 3 3 2 1	4 5 6 5 4	—	—	⇕	⇕	0
SUB (opr)	Subtract	ACCA ⬥ ACCA - M	IMM DIR EXT IX2 IX1 IX	A0 B0 C0 D0 E0 F0	ii dd hh ll ee ff ff	2 2 3 3 2 1	2 3 4 5 4 3	—	—	⇕	⇕	⇕
SWI	Software Interrupt	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 ⬥ $xFFC (vector PCL ⬥ $xFFD fetch)	INH	83		1	10	—	1	—	—	—
TAX	Transfer A to X	X ⬥ ACCA	INH	97		1	2	—	—	—	—	—
TST (opr) TSTA TSTX TST (opr) TST (opr)	Test for Negative or Zero	M - 0	DIR INH(A) INH(X) IX1 IX	3D 4D 5D 6D 7D	dd ff	2 1 1 2 1	4 3 3 5 4	—	—	⇕	⇕	0
TXA	Transfer X to A	ACCA ⬥ X	INH	9F		1	2	—	—	—	—	—
WAIT	Enable Interrupts, Halt CPU		INH	8F		1	2	—	0	—	—	—

3

■ 6367248 0149829 828 ■

3.4.7 Interrupts

Systems often require that normal processing be interrupted so that some external event may be serviced. The MC68HC705C8 may be interrupted by one of five different methods: any one of four maskable hardware interrupts (IRQ, SPI, SCI, or timer) and one nonmaskable software interrupt (SWI). Interrupts such as timer, SPI, and SCI have several flags which will cause the interrupt. Generally, interrupt flags are located in read-only status registers; their equivalent enable bits are located in associated control registers. The interrupt flags and enable bits are never contained in the same register. If the enable bit is a logic zero, it blocks the interrupt from occurring but does not inhibit the flag from being set. Reset clears all enable bits to preclude interrupts during the reset procedure.

The general sequence for clearing an 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. When any of these interrupts occur and the enable bit is a logic one, normal processing is suspended at the end of the current instruction execution.

Figure 3-8 shows how interrupts fit into the normal flow of CPU instructions. Interrupts cause the processor registers to be saved on the stack and the interrupt mask (I bit) to be set to prevent additional interrupts. The appropriate interrupt vector then points to the starting address of the interrupt service

3

M68HC05 Instruction Set Opcode Map

Opcode map. Each cell shows the mnemonic with its cycle count (shown here in parentheses). Columns are grouped by category and labeled with the high nibble (hex/binary); rows are the low nibble (hex/binary).

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

Abbreviations for Address Modes

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

Legend

MNEMONIC — BYTES — CYCLES

OPCODE IN HEXADECIMAL · OPCODE IN BINARY · ADDRESS MODE

6367248 0149831 486
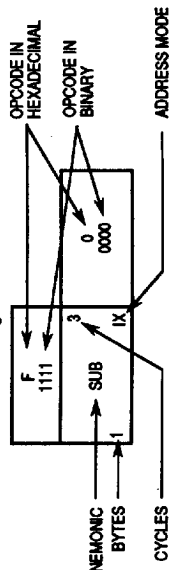
routine (refer to Figure 3-9 and Table 3-6 for vector location). Upon completion of the interrupt service routine, the RTI instruction (which is normally the last instruction of the routine) causes the register contents to be recovered from the stack followed by a return to normal processing.

NOTE

The interrupt mask bit (I bit) will be cleared if, and only if, the corresponding bit stored in the stack is zero.

Table 3-6. Vector Address for Interrupts and Reset

Register	Flag Name	Interrupts	CPU Interrupt	Vector Address
N A	N A	Reset	$\overline{\text{RESET}}$	$1FFE–$1FFF
N A	N A	Software	SWI	$1FFC–$1FFD
N A	N A	External Interrupt	$\overline{\text{IRQ}}$	$1FFA–$1FFB
Timer Status	ICF OFC TOF	Input Capture Output Compare Timer Overflow	TIMER	$1FF8–$1FF9
SCI Status	TDRE TC RDRF IDLE OR	Transmit Buffer Empty Transmit Complete Receiver Buffer Full Idle Line Detect Overrun	SCI	$1FF6–$1FF7
SPI Status	SPIF MODF	Transfer Complete Mode Fault	SPI	$1FF4–$1FF5

Reset and interrupt operations are often discussed together because they share the common concept of vector fetching to force a new starting point for further CPU operation. Unlike interrupts, there is no intention to ever return to whatever the CPU was doing before a reset occurred.

A low on the $\overline{\text{RESET}}$ input pin causes the program to vector to its starting address specified by the contents of memory location $1FFE and $1FFF. The I bit in the condition code register is also set. Much of the MCU is configured (forced) to a known state during reset.

3.4.7.1 SOFTWARE INTERRUPT (SWI). The software interrupt is an executable instruction. The action of the SWI instruction is similar to the hardware interrupts. The SWI is executed regardless of the state of the interrupt mask (I bit) in the condition code register. The interrupt service routine address is specified by the contents of memory location $1FFC and $1FFD.

6367248 0149832 312
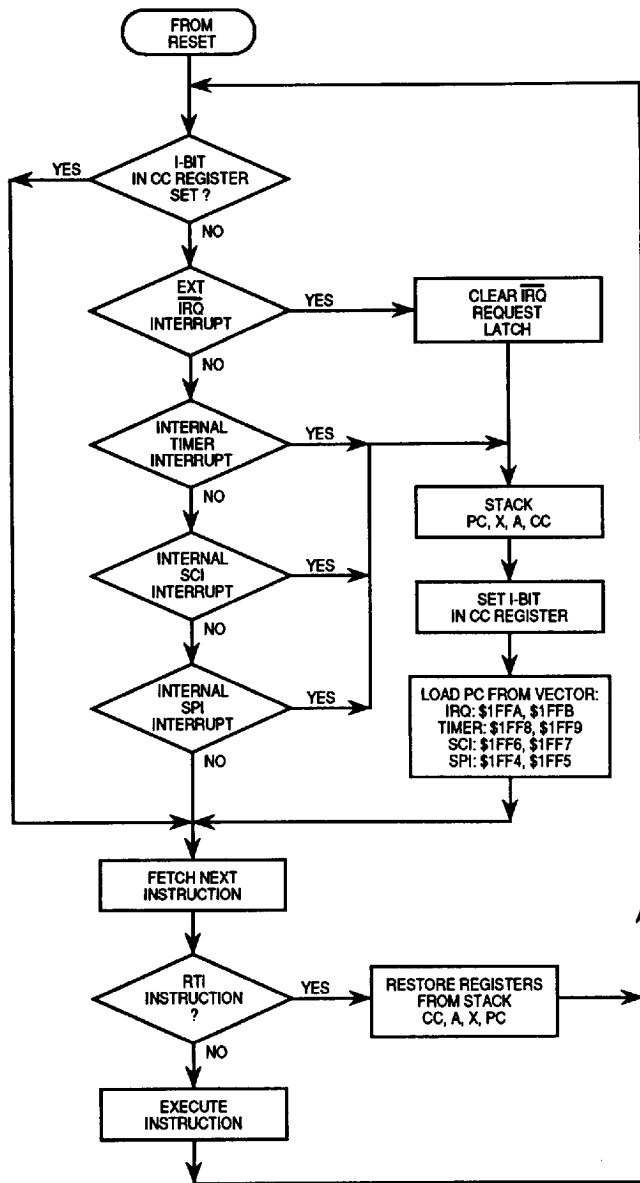
Figure 3-8. Hardware Interrupt Flowchart

■ 6367248 0149833 259 ■

TOWARD LOWER ADDRESSES
(LOWEST STACK ADDRESS IS $00C0)
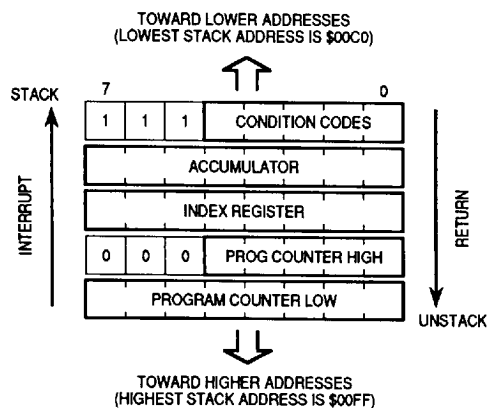
NOTE: When an interrupt occurs, CPU registers are saved on the stack in the order PCL, PCH, X, A, CC. On a return from interrupt registers are recovered from the stack in reverse order.

Figure 3-9. Interrupt Stacking Order

3.4.7.2 EXTERNAL INTERRUPT. If the interrupt mask (I bit) of the condition code register has been cleared and the external interrupt pin (IRQ) has gone low, then the external interrupt is recognized. When the interrupt is recognized, the current state of the CPU is pushed onto the stack and the I bit is set. This masks further interrupts until the present one is serviced. The interrupt service routine address is specified by the contents of memory location $1FFA and $1FFB.

The MC68HC705C8 MCU IRQ pin sensitivity is software programmable. Either negative edge- and level-sensitive triggering or negative edge-sensitive triggering are available. The MC68HC705C8 MCU uses the option register residing at location $1FDF to control the IRQ pin sensitivity.

3.4.7.3 TIMER INTERRUPT. There are three different interrupt flags that will cause a timer interrupt whenever they are set and enabled. These three interrupt flags are found in the three MSBs of the timer status register (TSR, location $13), and all three will vector to the same interrupt service routine ($1FF8–$1FF9).

■ 6367248 0149834 195 ■

All interrupt flags have corresponding enable bits (ICIE, OCIE, and TOIE) in the timer control register (TCR, location $12). 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 I bit in the condition code register is also cleared. 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 the associated control register.

3.4.7.4 SERIAL COMMUNICATIONS INTERFACE (SCI) INTERRUPT. An interrupt in the SCI 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 communication control register 2 (location $0F) is enabled. Software in the serial interrupt service routine must determine the priority and cause of the SCI interrupt by examining the interrupt flags and the status bits located in the serial communications status register (location $10) 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 the associated control register.

3.4.7.5 SERIAL PERIPHERAL INTERFACE (SPI) INTERRUPT. An interrupt in the SPI occurs when one of the interrupt flag bits in the serial peripheral status register (location $0B) is set, provided the I bit in the condition code register is clear and the enable bit in the serial peripheral control register (location $0A) is enabled. 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 the associated control register.

3.5 MICROCONTROLLER INPUT/OUTPUT

Since inputs to and outputs from the MCU are usually digital (0 to +5 Vdc at low power), interface logic is often needed to couple the MCU to external devices. Interface logic can operate in parallel or serial form.

Parallel interfaces allow I/O data transfer eight bits at a time, to parallel ports on the MCU. Serial interfaces transfer I/O data one bit at a time through a serial communications interface (SCI) or serial peripheral interface (SPI) that are parts of the MCU.

■ ■ 6367248 0149835 021 ■■

Data transfers between the MCU and external logic are controlled by the MCU.

NOTE

Tie all unused inputs and I/O ports to an appropriate logic level, either V_{DD} or V_{SS}.

3.5.1 Parallel I/O

The MC68HC705C8 MCU contains 31 general-purpose parallel I/O pins arranged in four ports. Ports A, B, and C are 8-bit ports in which the direction of each pin is programmable by software-accessible registers. Each 8-bit port has an associated 8-bit data direction register (DDR) as shown in Figures 3-10, 3-11, and 3-12.
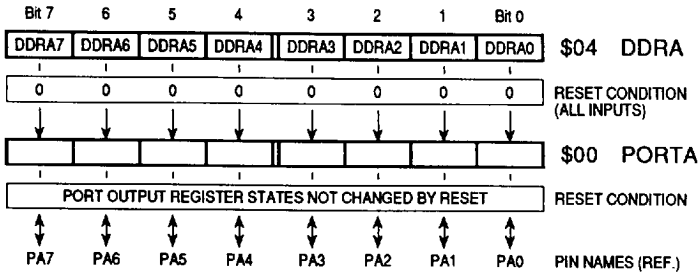
Figure 3-10. Port A and Data Direction A Registers
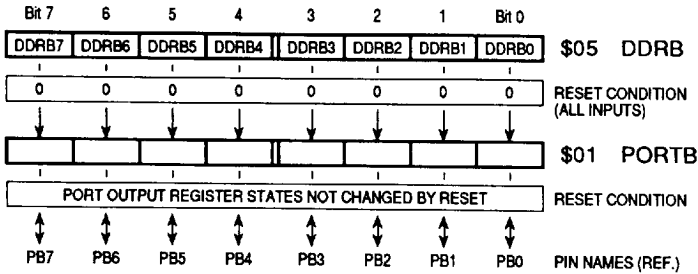
Figure 3-11. Port B and Data Direction B Registers

■ ᏝᏋᏝᎲᏋᏞᏝ ᎲᏝᏝᏌᏝᏝᏝᏝ ᎢᏝᏝ ■
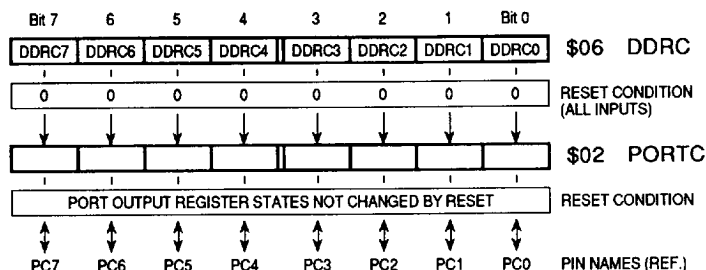
Figure 3-12. Port C and Data Direction C Registers

Any port A, B, or C pin is configured as an output if its corresponding DDR bit is set to a logic one. A pin is configured as an input if its corresponding DDR bit is cleared to a logic zero. At power-on or reset, all DDRs are cleared, which configure all port A, B, and C pins as inputs. The DDRs are capable of being written to or being read by the processor. Refer to Figure 3-13 and Table 3-7. When a port pin is configured as an output, a read of the data register actually reads the value of the output data latch and not the I/O pin.
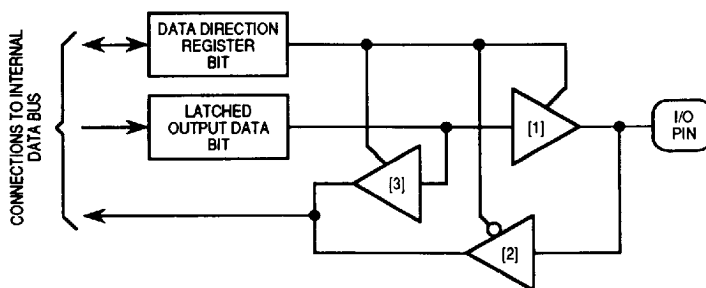
[1] - Output Buffer, enables latched output to drive pin when DDR bit is 1 (output)

[2] - Input Buffer, enabled when DDR bit is 0 (Input).

[3] - Input Buffer, enabled when DDR bit is 1 (Output).

Figure 3-13. Parallel Port I/O Circuitry

■ 6367248 0149837 9T4 ■

Table 3-7. I/O Pin Functions

R/W*	DDR	I/O Pin Function
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 output mode. The output data latch is read.

*R/W is an internal signal.

3.5.2 Serial I/O

Port D (see Figure 3-14) is a 7-bit fixed-direction input port. The SPI and SCI systems take control of port D pins when these systems are enabled. During power-on reset or external reset, all seven pins (PD5–PD0, PD7) are configured as input ports because all special-function output drivers are disabled. For example, with the SCI system enabled (RE = TE = 1), PD0 and PD1 inputs will read zero. With the SPI system disabled (SPE = 0), PD5–PD2 will read the state of the pin at the time of the read operation.

The SCI function uses two of the pins (PD1–PD0) for its receive data input (RDI) and transmit data output (TDO); the SPI function uses four of the pins (PD5–PD2) for its serial data input/output (MISO, MOSI), system clock (SCK), and slave select (SS), respectively.
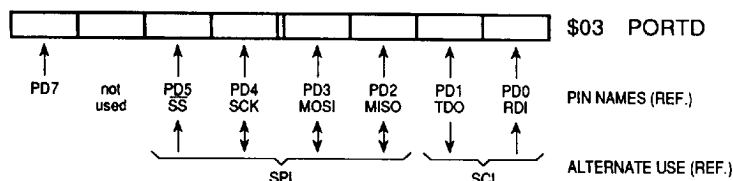
Figure 3-14. Port D Fixed Input Port

3.6 SERIAL COMMUNICATIONS INTERFACE (SCI)

SCI is one of two independent serial I/O subsystems in the MC68HC705C8. The other serial I/O system (called SPI) provides for high-speed synchronous serial communication to peripherals or other MCUs. The SCI is a full-duplex UART-type asynchronous system that can be used for communication between the MCU and a CRT terminal or a personal computer, or several widely

distributed MCUs can use their SCI subsystems to form a serial communications network.

The SCI uses standard nonreturn-to-zero (NRZ) format (one start bit, eight or nine data bits, and a stop bit). The most common data format is eight bits. An on-chip baud rate generator derives standard baud rate frequencies from the MCU oscillator. The SCI transmitter and receiver are functionally independent but use the same data format and baud rate. In this applications guide, "baud rate" and "bit rate" are used synonymously.

SCI Features:
- Two-Wire Serial Interface
- Standard NRZ (mark/space) Format
- Full-Duplex Operation (independent transmit and receive)
- Software Programmable for One of 32 Different Baud Rates
- Software-Selectable Word Length (8- or 9-bit words)
- Separate Transmitter and Receiver Enable Bits
- Communication may be Interrupt Driven

Receiver:
- Receiver Data Register Full Flag
- Error Detect Flags — Framing, Noise, Overrun
- Idle-Line Detect Flag
- Receiver Wakeup Function (idle or address bit)

Transmitter:
- Transmit Data Register Empty Flag
- Transmit Complete Flag (for modem control)
- Break Send

3.6.1 SCI Transmitter

The SCI transmitter block diagram is shown in Figure 3-15. The heart of the transmitter is the transmit serial shift register near the top of the figure. Usually, this shift register obtains its data from the write-only transmit buffer. Data is transferred into the transmit buffer when software writes to the SCI data register (SCDAT). Whenever data is transferred into the shifter from the transmit buffer, a zero is loaded into the LSB of the shifter to act as start bit, and a logic one is loaded into the last bit position to act as a stop bit. In the case of a preamble, the shifter is loaded with all ones, including the bit position usually holding the logic zero start bit. A preamble is loaded each time the transmit enable bit is written from zero to one. In the case of a send

■ 6367248 0149839 777 ■
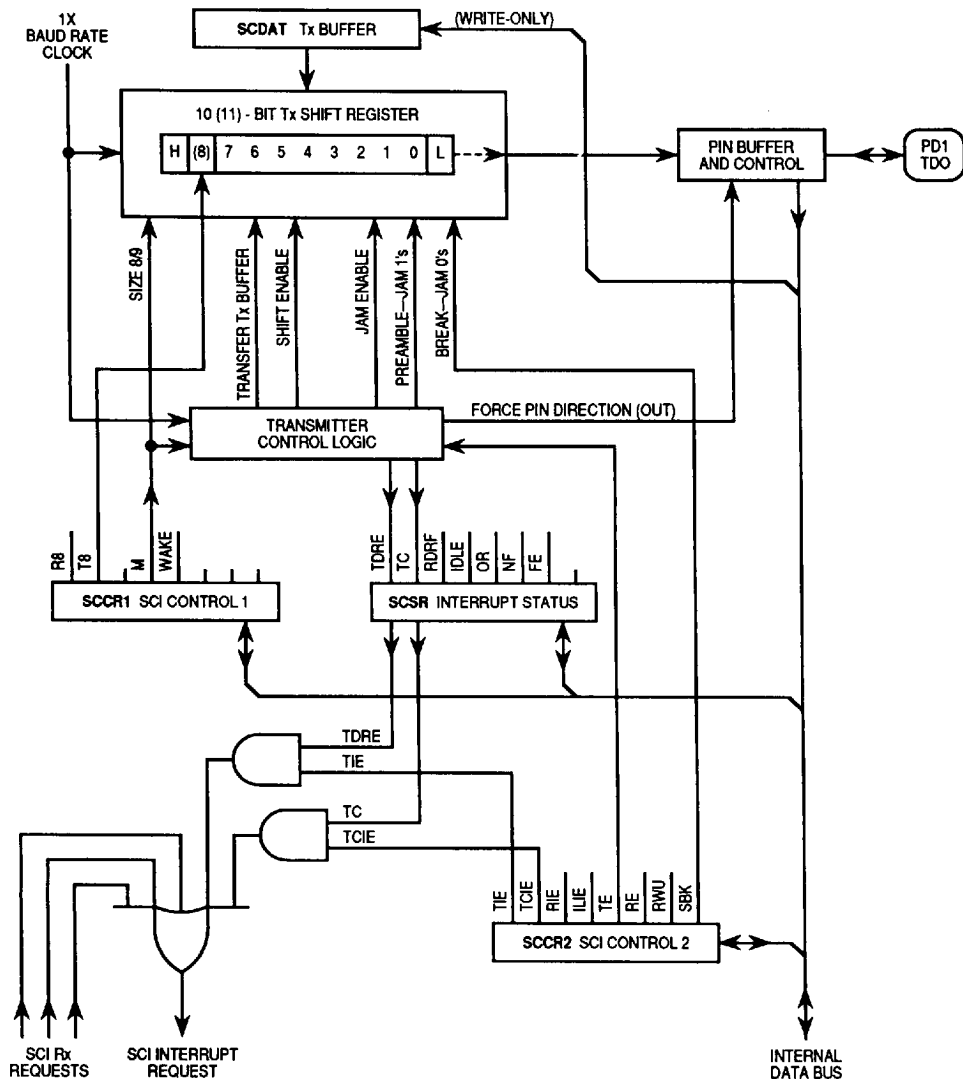
Figure 3-15. SCI Transmitter Block Diagram

break command, the shifter is loaded with all zeros, including the last bit position usually holding the logic one stop bit.

The T8 bit in SCI control register 1 (SCCR1) acts like an extra high-order bit (ninth bit) of the transmit buffer register. This ninth bit is only used if the M bit in SCCR1 is set, selecting the 9-bit data character format. The M bit also controls the length of idle and break characters.

The status flag and interrupt generation logic are shown in Figure 3-15. The transmit data register empty (TDRE) and transmit complete (TC) status flags in the SCI status register (SCSR) are automatically set by the transmitter logic. These two bits can be read at any time by software. The transmit interrupt enable (TIE) and transmit complete interrupt enable (TCIE) control bits enable the TDRE and TC flags, respectively, to generate SCI interrupt requests.

3.6.2 SCI Receiver

The receiver block diagram is shown in Figure 3-16. SCI received data comes in on the RDI pin, is buffered, and drives the data recovery block. The data recovery block is actually a high-speed shifter operating at 16 times the bit rate; the main receive serial shifter operates at one times the bit rate. This higher speed sample rate allows the start-bit leading edge to be located more accurately than a $1 \times$ clock would allow. The high-speed clock also allows several samples to be taken within a bit time so logic can make an intelligent decision about the logic sense of a bit (even in the presence of noise). The data recovery block provides the bit level to the main receiver shift register and also provides a noise flag status indication.

The heart of the receiver is the receive serial shift register. This register is enabled by the receive enable (RE) bit in the SCI control register 2 (SCCR2). The M bit from the SCCR1 register determines whether the shifter will be 10 or 11 bits. After detecting the stop bit of a character, the received data is transferred from the shifter to the SCDAT, and the receive data register full (RDRF) status flag is set. When a character is ready to be transferred to the receive buffer but the previous character has not yet been read, an overrun condition occurs. In the overrun condition, data is not transferred, and the overrun (OR) status flag is set to indicate the error.

There are three receiver-related interrupt sources in the SCI. These flags can be polled by software or, when enabled, cause an SCI interrupt request. The receive interrupt enable (RIE) control bit enables the RDRF and OR status
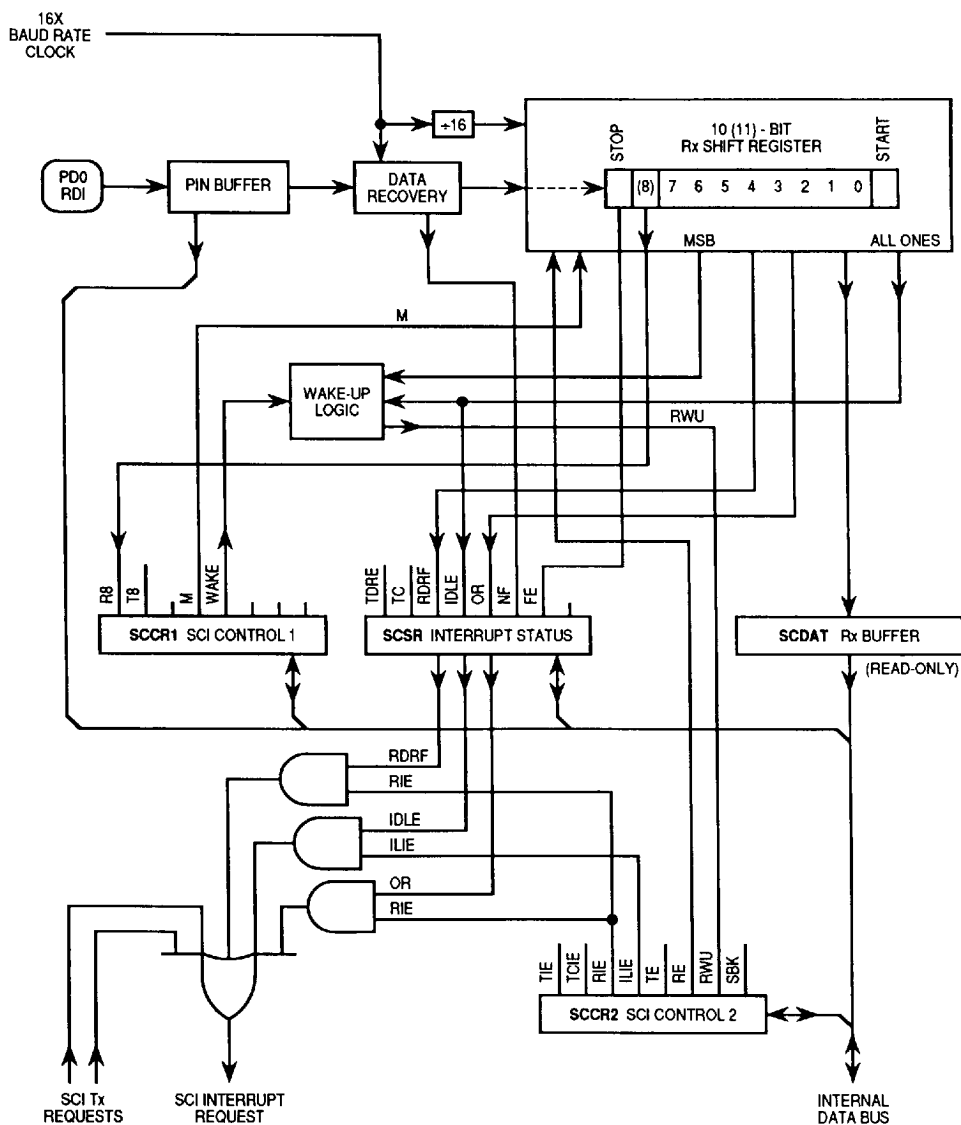
Figure 3-16. SCI Receiver Block Diagram

flags to generate hardware interrupt requests. The idle line interrupt enable (ILIE) control bit allows the IDLE status flag to generate interrupt requests.

3.6.3 Registers

The SCI system includes five registers (BAUD, SCCR1, SCCR2, SCSR, and SCDAT) and two external pins (TDO and RDI). When the SCI receiver and/or transmitter is enabled, the SCI logic takes control of the pin buffers for the associated port D pin(s). When the SCI is disabled, the TDO and RDI pins act as general-purpose inputs.

The main function of each of these registers will be discussed. Normally, the SCCR1, SCCR2, and BAUD registers would be written once to initialize and then not used again. An example of the software/programming procedure is shown later in this section.

3.6.3.1 BAUD RATE REGISTER (BAUD). The BAUD register (see Figure 3-17) is used to select the baud rate for the SCI system. Both the transmitter and receiver use the same data format and baud rate, which is derived from the MCU bus rate clock. The SCP1–SCP0 bits function as a prescaler for the SCR2–SCR0 bits. Together, these five bits provide multiple baud rate combinations for a given crystal frequency.

The diagram shown in Figure 3-18 and Tables 3-8 and 3-9 illustrate the divider chain used to obtain the baud rate clock (transmit clock). For example, using a 4-MHz crystal, the internal processor clock is 2 MHz.
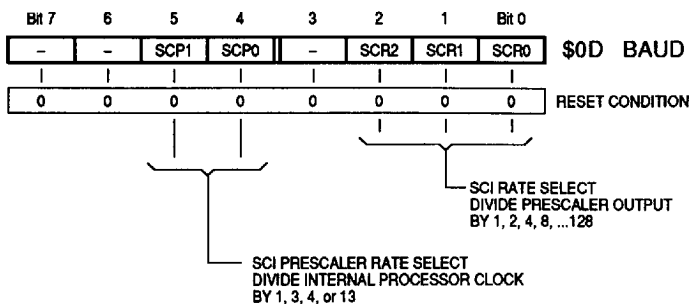
Figure 3-17. Baud Rate Register

■ ᏮᏴᏮᎪᎴᏴ 0149843 1T8 ■
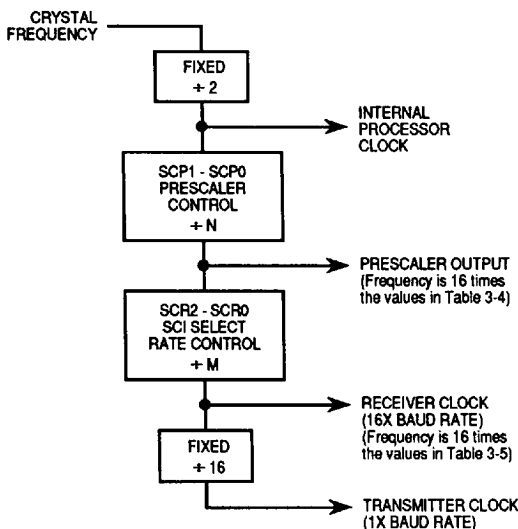
Figure 3-18. Rate Generator Division

Table 3-8. Prescaler 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	62.60 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 3-8 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 shown below for some representative prescaler outputs.

The SCP1–SCP0 bits in the baud rate register set the division factor (N in Figure 3-18) for the baud rate divider. Reset clears these bits, setting the prescaler to divide-by-one.

The SCR2, SCR1, and SCR0 bits are used to set the division factor (M in Figure 3-18) for the baud rate divider. Reset does not affect these bits.

Example:
From Table 3-8, find the crystal frequency used (in this case, 4 MHz). Next, find 9600 or a binary multiple of 9600. In this example, you would select

■ 6367248 0149844 034 ■

Table 3-9. Transmit Baud Rate Output

SCR Bits			Divided By	Representative Highest Prescaler Baud Rate Output				
2	1	0		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 3-9 illustrates how the SCI select bits can be used to provide lower transmitter baud rate 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 receive clock is 16 times higher in frequency than the actual baud rate.

the bottom row which corresponds to SCP1:SCP0 = 1:1 (divide-by-thirteen). Next, find the column in Table 3-5 that corresponds to 9600 Hz. Find the desired baud rate in this column. In this example, you would select the top row, which corresponds to SCR2:SCR1:SCR0 = 0:0:0 (divide-by-one).

3.6.3.2 SERIAL COMMUNICATIONS CONTROL REGISTER ONE (SCCR1). The serial communications control register one (SCCR1) shown in Figure 3-19 includes three bits associated with the optional 9-bit data format. The WAKE bit is used to select one of two methods of receiver wakeup. Normal setup for bit M is 0 for 8-bit words. The other register bits are not used in most systems. In a typical system, this register would be written to $00 during initialization.
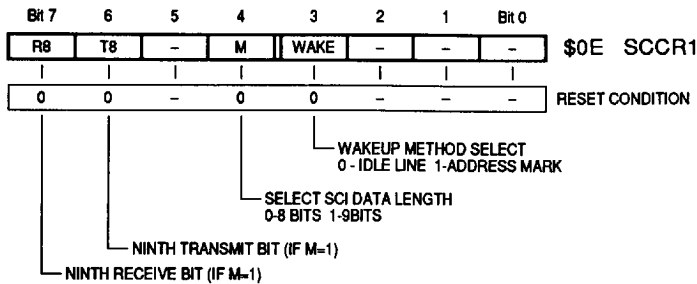
Figure 3-19. Serial Communications Control Register One

■ 6367248 0149845 T70 ■

This Material Copyrighted By Its Respective Manufacturer

3.6.3.3 SERIAL COMMUNICATIONS CONTROL REGISTER TWO (SCCR2). The serial communications control register two (SCCR2) shown in Figure 3-20 is the main control register for the SCI subsystem. This register can enable/disable the transmitter or receiver, enable the system interrupts, and provide the wakeup enable bit and a "send break code" bit. The TIE, TCIE, RIE, and ILIE bits are local interrupt enable controls, which determine whether SCI status flags will be polled or generate hardware interrupt requests.
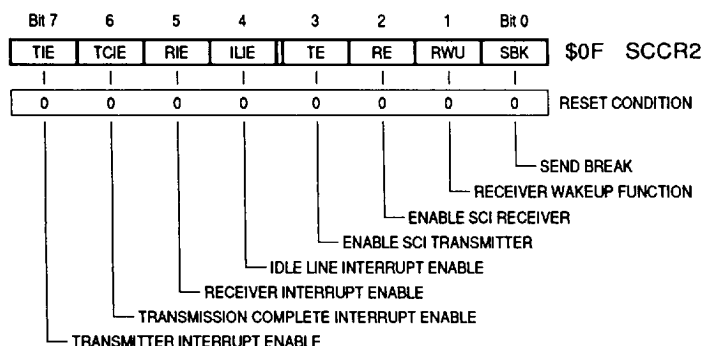
Figure 3-20. Serial Communications Control Register Two

In a typical system:
  TE and RE would be written to one to enable the transmitter and receiver subsystems.

  ILIE, RWU, and SBK would seldom be used and would be written to zero.

  If interrupts were not being used, TIE, TCIE, and RIE would be written to zero. If interrupts were used, these three bits would be written to one.

For example, in a system which does not use interrupts, SCCR2 would be loaded with $0C during initialization.

3.6.3.4 SERIAL COMMUNICATIONS STATUS REGISTER (SCSR).

The SCI status register (SCSR) in Figure 3-21 contains two transmitter status flags and five receiver related status flags. The TDRE and RDRF bits are always used. The TC and IDLE bits are not commonly used.
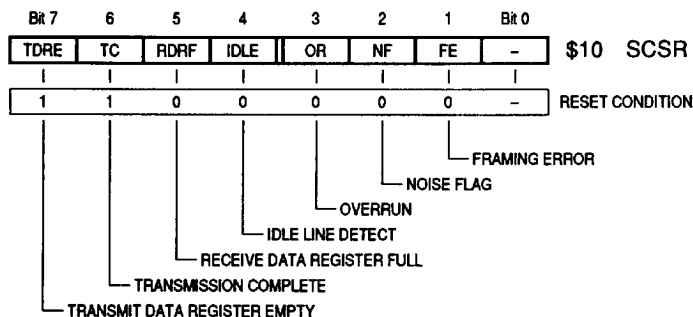
Figure 3-21. Serial Comunications Status Register

The OR, NF, and FE bits should be monitored and may or may not be used, depending on the type of SCI system. For errors to be corrected, both the transmitting and receiving device must have a common method of handling errors.

There are two major types of communication links associated with the SCI. An example of a direct connection would be an MCU connected to a personal computer. In this direct connection link OR, NF, and FE errors are very unlikely and are typically ignored. The second type of link involves two remote devices where each is connected to a modem. In this type of link, errors are more likely and both computers would typically use a protocol that permits re-transmission when an error is detected.

3.6.3.5 SERIAL COMMUNICATIONS DATA REGISTER (SCDAT). The SCI SCDAT data register (see Figure 3-22) has two functions: it is the transmit data register when written to and the receive data register when read. Both the transmitter and receiver are double buffered (see Figure 3-23), so back-to-back characters can be handled easily even if the CPU is delayed in responding to the completion of an individual character.
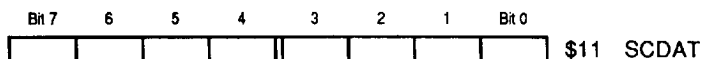
Figure 3-22. Serial Communications Data Register
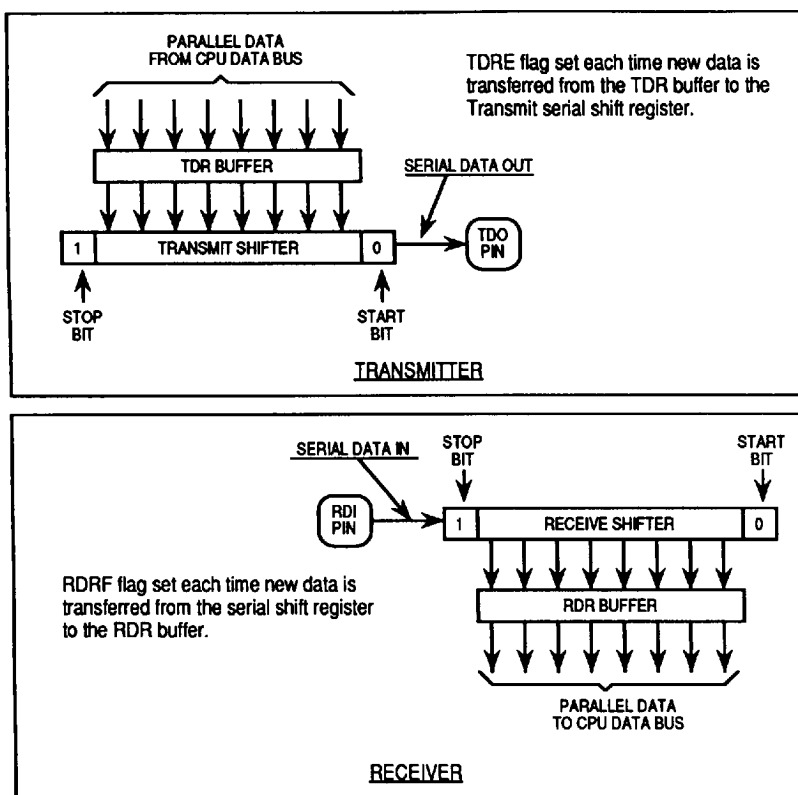
Figure 3-23. Double Buffering

■ 6367248 0149848 78T ■

3.6.4 Data Formats

The standard NRZ data formats used for communications are shown in Figure 3-24. The upper portion of this figure shows the normal 8-bit data format; the lower portion of the figure shows the 9-bit data format. The 9-bit data format is selected by setting the M control bit in SCCR1 to 1.

The basic characteristics of the NRZ format are as follows:

1) A high level indicates a logic one and a low level, a logic zero.

2) The idle line is high prior to message transmission/reception.

3) A start bit (logic zero) is transmitted/received as the first bit of data in a character.

4) Data is transmitted/received LSB first.

5) The last bit in a character (bit 10 or 11) is a high (stop bit).

6) A break is a low (logic zero) for 10 or 11 bit times.

[1] - Control bit "M" selects optional ninth data bit.
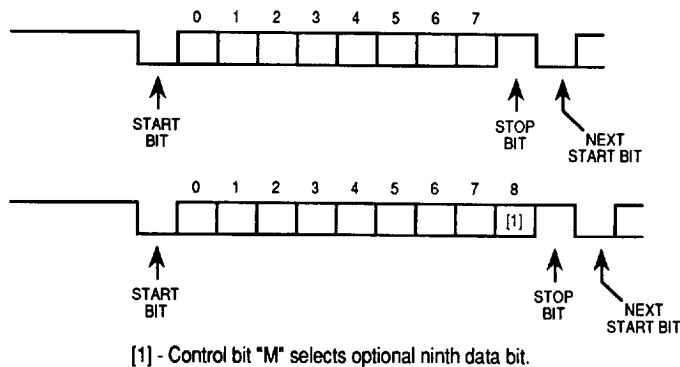
Figure 3-24. Data Formats

3.6.5 Hardware Procedures

Some simple hardware setup is required. A universal standard RS232 cable is used to interconnect the SCI to a CRT terminal or the PC. The user would usually have to provide an external level shifter buffer (MC145406) to convert the RS232 (typically ±12 volts) to the 0–5 volt logic levels used by the MC68HC705C8.

3.6.6 Software Procedures

The following paragraphs and flowcharts discuss software procedures. These flowcharts illustrate how straightforward normal SCI operations are.

3.6.6.1 INITIALIZATION PROCEDURE.   The following list reflects the initialization procedure.

1) Write to BAUD register (SCP1–SCP0, SCR2–SCR0) to set baud rate.

2) Write to SCCR1 (R8, T8, M, WAKE) to set character length and choose wakeup method.

3) Write to SCCR2 (TIE, TCIE, RIE, ILIE, TE, RE, RWU, SBK) to enable desired interrupt sources. To turn on the transmitter and receiver, RWU and SBK would be written to zero during initialization.

The following is a reference list of interrupt enable control bits versus the interrupt source(s) they enable:

Enable	Flags	Interrupt Source Names
TIE	TDRE	Transmit data register empty
TCIE	TC	Transmit complete
RIE	RDRF, OR	Receive data register full, overrun
ILIE	IDLE	Idle line detect

■ 6367248 0149850 338 ■

3.6.6.2 NORMAL TRANSMIT OPERATION. Refer to Figure 3-25, a flowchart of the normal transmit operation.

FLOWCHART   MNEMONIC PROGRAM

```
        ┌─────────────┐
        │    START    │
        │  SUBROUTINE │
        └──────┬──────┘
               │
    ┌──────────▼──────────┐
 NO │      TDRE = 1       │    SENDATA    BRCLR 7,SCSR,SENDATA
    │         ?           │
    └──────────┬──────────┘
            YES│
        ┌──────▼──────┐
        │ WRITE DATA  │─────────────── STA    SCDAT
        │  TO SCDAT   │
        └──────┬──────┘
        ┌──────▼──────┐
        │ RETURN FROM │─────────────── RTS
        │  SUBROUTINE │
        └─────────────┘
```
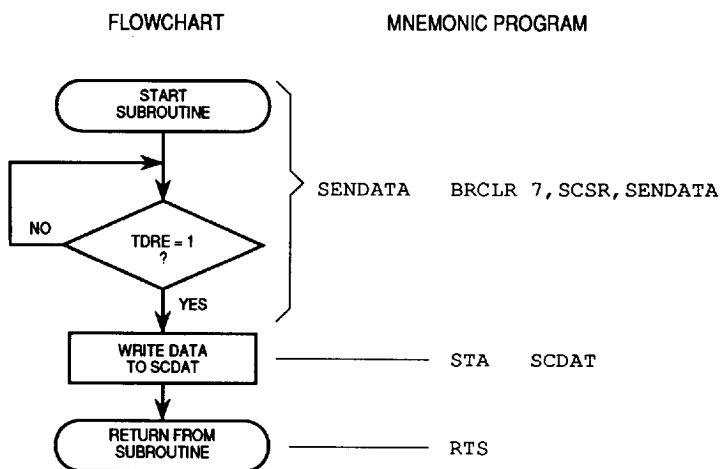
Figure 3-25. SCI Normal Transmit Operation Flowchart

3.6.6.3 NORMAL RECEIVE OPERATION. Refer to Figure 3-26, a flowchart of the normal receive operation.

FLOWCHART   MNEMONIC PROGRAM

```
        ┌─────────────┐
        │    START    │
        │  SUBROUTINE │
        └──────┬──────┘
               │
    ┌──────────▼──────────┐
 NO │      RDRF = 1       │    GETDATA    BRCLR 5,SCSR,GETDATA
    │         ?           │
    └──────────┬──────────┘
            YES│
        ┌──────▼──────┐
        │  READ DATA  │─────────────── LDA    SCDAT
        │ FROM SCDAT  │
        └──────┬──────┘
        ┌──────▼──────┐
        │ RETURN FROM │─────────────── RTS
        │  SUBROUTINE │
        └─────────────┘
```
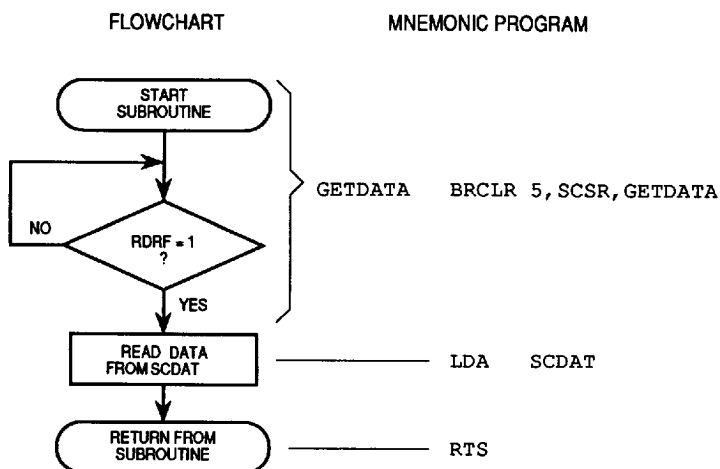
Figure 3-26. SCI Normal Receive Operation Flowchart

■ 6367248 0149851 274 ■

3.6.7 SCI Application Example

Figure 3-27 is an example software program for communication between the SCI of the MCU and a dumb terminal. The MCU will receive (read) an ASCII character that was sent by the dumb terminal. The MCU will then translate the 8-bit binary character representing the ASCII character into two ASCII characters.

When this translation is completed, the MCU will transmit a <CR>, line feed, a $ sign and the two characters that represent the original hexadecimal equivalent of the received character back to the terminal. The program then waits for another character.

In practice, the following would occur:

You type a number/character on the keyboard. It goes from the terminal to the MCU over the SCI receiver. Use the example of the letter "A".

The program translates "A" to "4" and "1", then sends CR, line feed, $, 4, and 1, to the SCI transmitter.

When the transmission is complete, the program goes back to the top for another keyboard number/character to be sent over the SCI receiver.

Table 3-10 is a chart of the ASCII-hexadecimal code conversion.

Table 3-10. ASCII-Hexadecimal Code Conversion

ASCII CHARACTER SET (7-BIT CODE)								
LS Dig. \ MS Dig.	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

6367248 0149852 100

```
         ************************************************
         * Simple 68HC05 SCI Program Example          *
         ************************************************

000d              BRATE   EQU   $0D      -,-,SCP1,SCP0;-,SCR2,SCR1,SCR0
000e              SCCR1   EQU   $0E      R8,T8,-,M;WAKE,-,-,-
000f              SCCR2   EQU   $0F      TIE,TCIE,RIE,ILIE;TE,RE,RWU,SBK
0011              SCDAT   EQU   $11      Read - RDR; Write - TDR
0010              SCSR    EQU   $10      TDRE,TC,RDRF,IDLE;OR,NF,FE,-

00a0              TEMP    EQU   $A0      One byte temp storage location
00a1              TEMPHI  EQU   $A1      Upper byte changed to ASCII
00a2              TEMPLO  EQU   $A2      Lower byte changed to ASCII

0500                      ORG   $500     Program will start at $0500

0500 a6 30    INITIAL LDA   #%00110000  Begin initialization
0502 b7 0d            STA   BRATE       Baud rate to 4800 @2MHz Xtal
0504 a6 00            LDA   #%00000000  Set up SCCR1
0506 b7 0e            STA   SCCR1       Store in SCCR1 register
0508 a6 0c            LDA   #%00001100  Set up SCCR2
050a b7 0f            STA   SCCR2       Store in SCCR2 register
050c cd 05 43 START   JSR   GETDATA     Checks for receive data
050f b7 a0            STA   TEMP        Store received ASCII data in temp
0511 a4 0f            AND   #$0F        Convert LSB of ASCII char to hex
0513 aa 30            ORA   #$30        $3(LSB) = "LSB"
0515 a1 39            CMP   #$39        3A-3F need to change to 41-46
0517 23 02            BLS   ARN1        Branch if 30-39 OK
0519 ab 07            ADD   #7          Add offset
051b b7 a2    ARN1    STA   TEMPLO      Store LSB of hex in TEMPLO
051d b6 a0            LDA   TEMP        Read the original ASCII data
051f 44               LSRA              Shift right 4 bits
0520 44               LSRA
0521 44               LSRA
0522 44               LSRA
0523 aa 30            ORA   #$30        ASCII for N is $3N (N=0-9)
0525 a1 39            CMP   #$39        3A-3F need to change to 41-46
0527 23 02            BLS   ARN2        Branch if 30-39
0529 ab 07            ADD   #7          Add offset
052b b7 a1    ARN2    STA   TEMPHI      MS nibble of hex to TEMPHI
052d a6 0d            LDA   #$0D        Load hex value for "<CR>"
052f ad 18            BSR   SENDATA     Carriage return
0531 a6 0a            LDA   #$0A        Load hex value for "<LF>"
0533 ad 14            BSR   SENDATA     Line feed
0535 a6 24            LDA   #'$         Load hex value for "$"
0537 ad 10            BSR   SENDATA     Print dollar sign
0539 b6 a1            LDA   TEMPHI      Get high half of hex value
053b ad 0c            BSR   SENDATA     Print
053d b6 a2            LDA   TEMPLO      Get low half of hex value
053f ad 08            BSR   SENDATA     Print
0541 20 c9            BRA   START       Branch back to start
```

Figure 3-27. SCI Application Example Program (Sheet 1 of 2)

■ 6367248 0149853 047 ■

```
                    *** Get an SCI character, return w/ it in A
0543 0b 10 fd GETDATA  BRCLR 5,SCSR,GETDATA RDRF = 1 ?
0546 b6 11             LDA   SCDAT         OK, get
0548 81               RTS                  ** Return from GETDATA **

                    *** Send an SCI character, call sub w/ it in A
0549 0f 10 fd SENDATA  BRCLR 7,SCSR,SENDATA TDRE = 1 ?
054c b7 11             STA   SCDAT         OK, send
054e 81               RTS                  ** Return from SENDATA **
```

Figure 3-27. SCI Application Example Program (Sheet 2 of 2)

3.7 SYNCHRONOUS SERIAL PERIPHERAL INTERFACE (SPI)

The SPI subsystem included in the MC68HC705C8 allows the MCU to communicate with peripheral devices. Peripheral devices can be as simple as an ordinary TTL shift register or as complex as a complete subsystem such as an LCD display driver or an A/D converter subsystem. The SPI system is flexible enough to interface directly with numerous standard product peripherals from several manufacturers.

SPI is an added feature for those applications that require more inputs and outputs than there are parallel I/O pins on the MCU. SPI offers a very easy way to expand the I/O function while using a minimum number of MCU pins. The SPI block diagram is shown in Figure 3-28.

SPI features are as follows:
- 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

An SPI subsystem can operate under software control in either complex or simple system configurations:
- One Master MCU and Several Slave MCUs
- Several MCUs Interconnected in a Multimaster System
- One Master MCU and One or More Slave Peripherals

6367248 0149854 T83
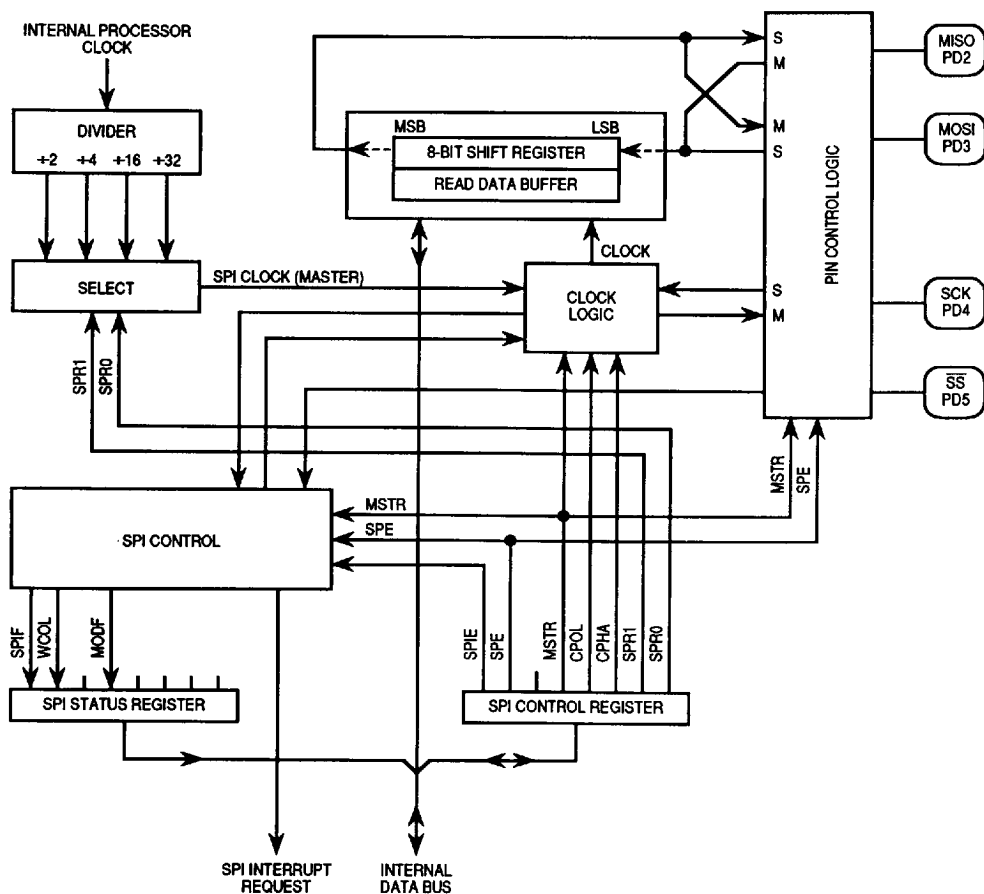
Figure 3-28. SPI Block Diagram

■ 6367248 0149855 91T ■

The majority of all applications use one MCU device as the master. This master initiates and controls the transfer of data to/from one or more slave peripheral devices that receive/supply the data being transferred. Slaves can read data from or transfer data to the master only after the master instructs an action to occur. This system configuration will be discussed in this applications guide.

3.7.1 Data Movement

There is no need to specify the direction of data movement for each transfer because the master simultaneously transmits and receives serial data on separate pins every transfer. When an SPI transfer occurs, an 8-bit character is shifted out on one data pin while a different 8-bit character is simultaneously shifted in on a second data pin (see Figure 3-29). Another way to think of this is that an 8-bit shift register in the master and another in the slave are connected as a circular 16-bit shift register. When a transfer occurs, this distributed shift register is shifted eight bit positions so the characters in the master and slave are effectively exchanged.

Many simple slave devices are designed to only receive data from a master or only supply data to a master. For example, a serial-to-parallel shift register can act as an 8-bit output port. An MCU configured as a master SPI device would initiate a transfer to send an 8-bit data value to the shift register. Since the shift register does not send any data to the master, the master would simply ignore whatever it received as a result of that transmission.
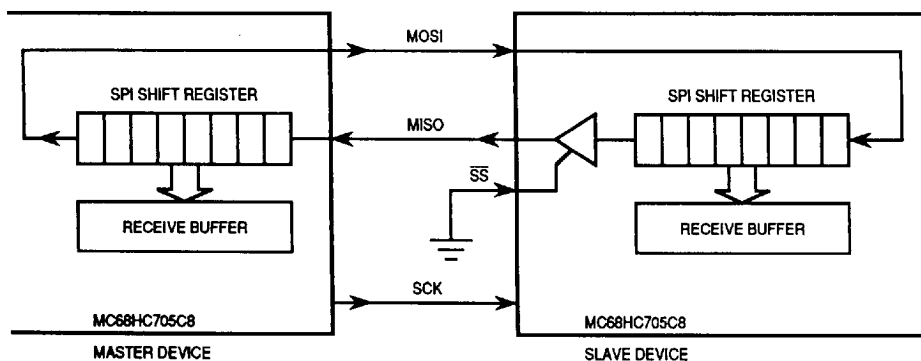
Figure 3-29. Shift Register Operation

3.7.2 Functional Description

Four I/O pins located at port D are associated with SPI data transfers. They are the serial clock (SCK–PD4), the master in/slave out (MISO–PD2) data line, the master out/slave in (MOSI–PD3) data line, and the active-low slave select ($\overline{SS}$–PD5). When the SPI system is not utilized, the four pins ($\overline{SS}$, SCK, MISO, and MOSI) are configured as general-purpose inputs (PD5, PD4, PD3, and PD2).

In a master configuration, the master start logic receives an input from the CPU (in the form of a write to the SPI data register) and originates the serial clock (SCK) based on the internal processor clock. This clock is also used internally to control the state controller as well as the 8-bit shift register. Data is parallel loaded into the 8-bit shift register (during the CPU write to SPDR) and then shifted out serially to the MOSI pin for application to the serial input line of the slave device(s). At the same time, data is applied serially from a slave device through the MISO pin to the 8-bit shift register. After the eighth shift in a transfer, data is parallel transferred to the read buffer where it is available to the internal data bus during a CPU read cycle. The SPIF status flag is used by the master and slave devices to indicate when a transfer is complete.

3.7.3 Pin Descriptions

The four I/O pins are discussed in the following paragraphs.

3.7.3.1 SERIAL DATA PINS (MISO, MOSI).

The master-in slave-out (MISO) and master-out slave-in (MOSI) data pins are used for transmitting and receiving data serially: MSB first, LSB last. When the SPI is configured as a master, MISO is the master data input line and MOSI is the master data output line. In the master device, the MSTR control bit (bit 4 of the serial peripheral control register) is set to a logic one (by the program) to allow the master device to output data on its MOSI pin. When the SPI is configured as a slave, these pins reverse roles; MISO becomes the slave data output line and MOSI becomes the slave data input line.

The timing diagram of Figure 3-30 shows the relationship between data and clock (SCK). As shown in Figure 3-30, four possible timing relationships may be chosen by using control bits CPOL and CPHA. Setting CPOL is equivalent to putting an inverter in series with the clock signal. CPHA selects one of two fundamentally different clocking protocols to allow the SPI system to communicate with virtually any synchronous serial peripheral device.

■ 6367248 0149857 792 ■
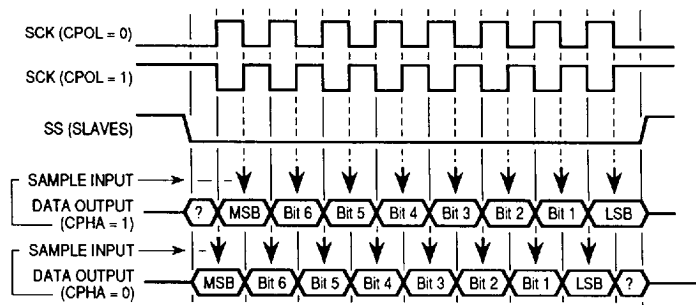
Figure 3-30. Data/Clock Timing Diagram

3.7.3.2 SERIAL CLOCK (SCK). SCK is used to synchronize the movement of data both in and out of the device through the MOSI and MISO pins. The SCK pin is an output when the SPI is configured as a master and an input when the SPI is configured as a slave.

When the SPI is configured as a master, the SCK signal is derived from the internal MCU bus clock. When the master initiates a transfer, eight clock cycles are automatically generated on the SCK pin. In both the master and slave SPI devices, data is shifted on one edge of the SCK signal and sampled on the opposite edge, where data is stable. Two bits (SPR0 and SPR1) in the SPCR (location $0A) of the master device select the clock rate. Both master and slave devices must be programmed to similar timing modes for proper data transfers, as controlled by the CPOL and CPHA bits in the SPCR.

3.7.3.3 SLAVE SELECT ($\overline{\text{SS}}$). The $\overline{\text{SS}}$ pin behaves differently on master devices than on slave devices. On a slave, this pin is used to enable the SPI slave for a transfer. On a master, the $\overline{\text{SS}}$ pin is normally pulled high externally.

3.7.4 Registers

Three registers in the SPI provide control, status, and data storage functions. These registers include the serial peripheral control register (location $0A), serial peripheral status register (location $0B), and serial peripheral data I/O register (location $0C).

■ 6367248 0149858 629 ■

3.7.4.1 SERIAL PERIPHERAL CONTROL REGISTER (SPCR).
In most systems, this register (Figure 3-31) is written only once shortly after reset to initialize the SPI system.
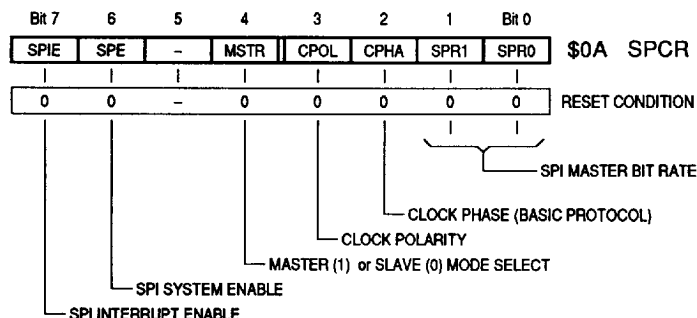
Figure 3-31. Serial Peripheral Control Register

The SPCR bits have the following functions:

SPIE
  0 = SPI interrupts are disabled (the most common configuration).
  1 = SPI interrupt requests are enabled if SPIF and/or MODF is set to one.

SPE
  0 = SPI system is turned off.
  1 = SPI system is turned on.

MSTR
  0 = SPI is configured as a slave.
  1 = SPI is configured as a master.

CPOL
  0 = Active-high clocks selected, SCK idles low.
  1 = Active-low clocks selected, SCK idles high.
  (This bit is used in conjunction with the clock phase control bit to produce the desired clock-data relationship between master and slave.)

CPHA
  The clock phase bit, in conjunction with the CPOL bit, controls the relationship between the data on the MISO and MOSI pins and the clock produced or received at the SCK pin. CPHA selects one of two fundamentally

■ 6367248 0149859 565 ■

different clocking protocols to allow the SPI system to communicate with virtually any synchronous serial peripheral device.

SPR1/SPR0
These two serial peripheral rate bits select one of four bit rates to be used as SCK if the device is a master; they have no effect in the slave mode.

SPR1	SPR0	Internal Processor Clock Divided By	Frequency if XTAL is 4.0 MHz	Frequency if XTAL is 2 MHz
0	0	2	1.0 MHz	500.0 kHz
0	1	4	500.0 kHz	250.0 kHz
1	0	16	125.0 kHz	62.50 kHz
1	1	32	62.5 kHz	31.25 kHz

3.7.4.2 SERIAL PERIPHERAL STATUS REGISTER (SPSR). This read-only register (Figure 3-32) contains status flags which indicate the completion of an SPI transfer and the occurrence of certain SPI system errors. The flags are automatically set by the SPI events; the flags are cleared by automatic software sequences and upon reset. In the majority of all systems, only the SPIF status bit is important.
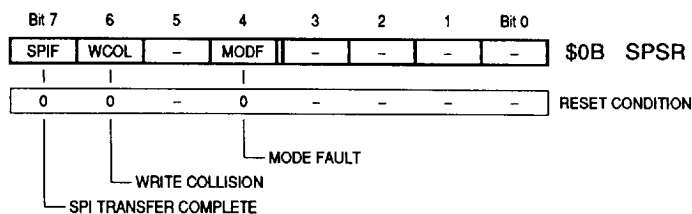
Figure 3-32. Serial Peripheral Status Register

The bits in this register have the following functions:

SPIF
When set to one, the serial peripheral data transfer flag bit notifies the user that a data transfer between the MCU and an external peripheral device has been completed. The transfer of data is initiated by the master device writing to its serial peripheral data register. SPIF is automatically cleared by reading SPSR with SPIF set, followed by an access of the SPI data register.

WCOL

> The write-collision status bit notifies the user that an attempt was made to write to the serial peripheral data register while a data transfer with an external peripheral device was in progress. The transfer continues uninterrupted, and the write will be unsuccessful.

MODF

> This flag is set if the $\overline{SS}$ signal goes to its active-low level while the SPI is configured as a master (MSTR = 1). In normal systems, this would never be possible. For information on how to use MODF in multimaster systems, see BR594/D, the *MC68HC705C8 Technical Summary*.

3.7.4.3 SERIAL PERIPHERAL DATA I/O REGISTER (SPDR). The SPDR (Figure 3-33) in the master MCU device is used to transmit data to and receive data from the slave device. Only a write to this register in a master will initiate transmission/reception of data. The data is then loaded directly into the 8-bit shift register where eight bits are shifted out on the MOSI pin to the slave while another eight bits are simultaneously shifted in on the MISO pin to the 8-bit shift register. At the completion of data transmission, the SPIF status bit is set. A write or read of the SPDR, after reading SPSR with SPIF set, will clear SPIF.

Bit 7	6	5	4	3	2	1	Bit 0	
								$0C SPDR
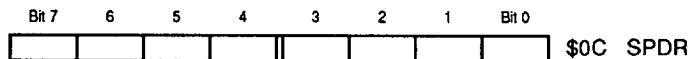
Figure 3-33. Serial Peripheral Data I/O Register

3.7.5 SPI Application Example

The example application and program are similar to the one shown in Section 2, paragraph 2.5, except the SPI function will be added.

A switch is connected to an input pin. When the switch is closed, the program will send data out to a peripheral device using the SPI function and will cause an LED connected to an output pin to light for about one second and then go out.

The peripheral device used in this application is an MC74HC595 serial-to-parallel shift register. Hardware setup, the SPI control register, and the software program will be discussed briefly.

Figure 3-34 shows the hardware connections for the SPI application example. The SPI signals at the left of the diagram come from the PGMR board (an M68HC05 PGMR, available from a Motorola distributor) or directly from the MC68HC705C8. The shift register outputs (QA–QH of the MC74HC595) will be monitored with an oscilloscope. In this example, the MISO line is not used. The shifter is selected by the general-purpose output PC3 (but could have been driven by any general-purpose output). The $\overline{SS}$ pin of the MC68HC705C8 is an input in master mode and must be tied high.
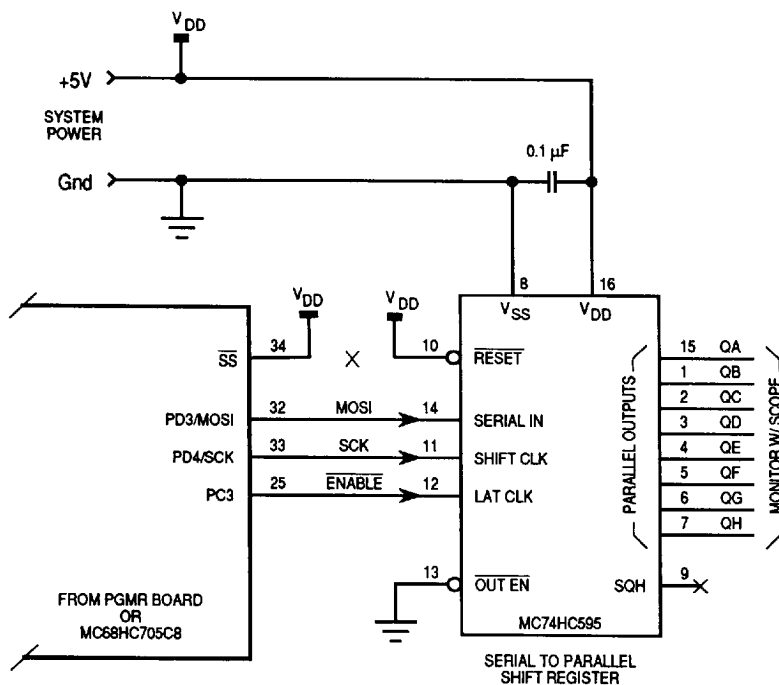
Figure 3-34. SPI Application Example Diagram

■ 6367248 0149862 05T ■

This Material Copyrighted By Its Respective Manufacturer

To initialize the SPI function, the SPCR (SPIE, SPE, —, MSTR, CPOL, CPHA, SPR1, SPR0) bits need to be written. For this application, the SPCR was initialized with %01010000 or $50.

SPIE = 0   No interrupts involved in this application.
SPE = 1   Enable the SPI system.
— = 0   Don't care bit.
MSTR = 1   MC68HC705C8 is the master.
CPOL = 0   Selects clock rest at low value.
CPHA = 0   MC74HC595 accepts data at rising clock edge
SPR1 = 0   Internal processor clock divide by two.
SPR0 = 0   (Shift rate = 500 kHz for a 2-MHz crystal).

The SPCR needs to be initialized once. For each transfer, there is a four-step sequence:

1) Enable the slave. In this example the PC3 general-purpose output provides the enable signal to the MC74HC595 peripheral.

2) Write data to SPDR to initiate the transfer.

3) Wait for SPIF. The slave cannot be disabled until the transfer is finished.

4) Disable the slave.

The flowchart and mnemonics for the SPI application example are shown in Figure 3-35.

Assume this application program has been assembled and downloaded to an MC68HC705C8. You can test this program by using an oscilloscope connected to the MC74HC595 parallel data outputs (pins 1–7 and 15). The program is arranged to increment the 8-bit parallel bit value each time the switch is pressed. Figure 3-36 is the complete listing for the SPI application example program.

3.8 PROGRAMMABLE TIMER

The programmable timer can be used for many purposes, including input waveform measurements, while simultaneously generating an output waveform. The architecture of the main timer is primarily a software driven system. Software can be written for measuring pulse widths and frequencies, for controlling timer output signals, or for timing delays.

The programmable timer is based on a 16-bit free-running counter preceded by a prescaler that divides the internal processor clock by four. A timer
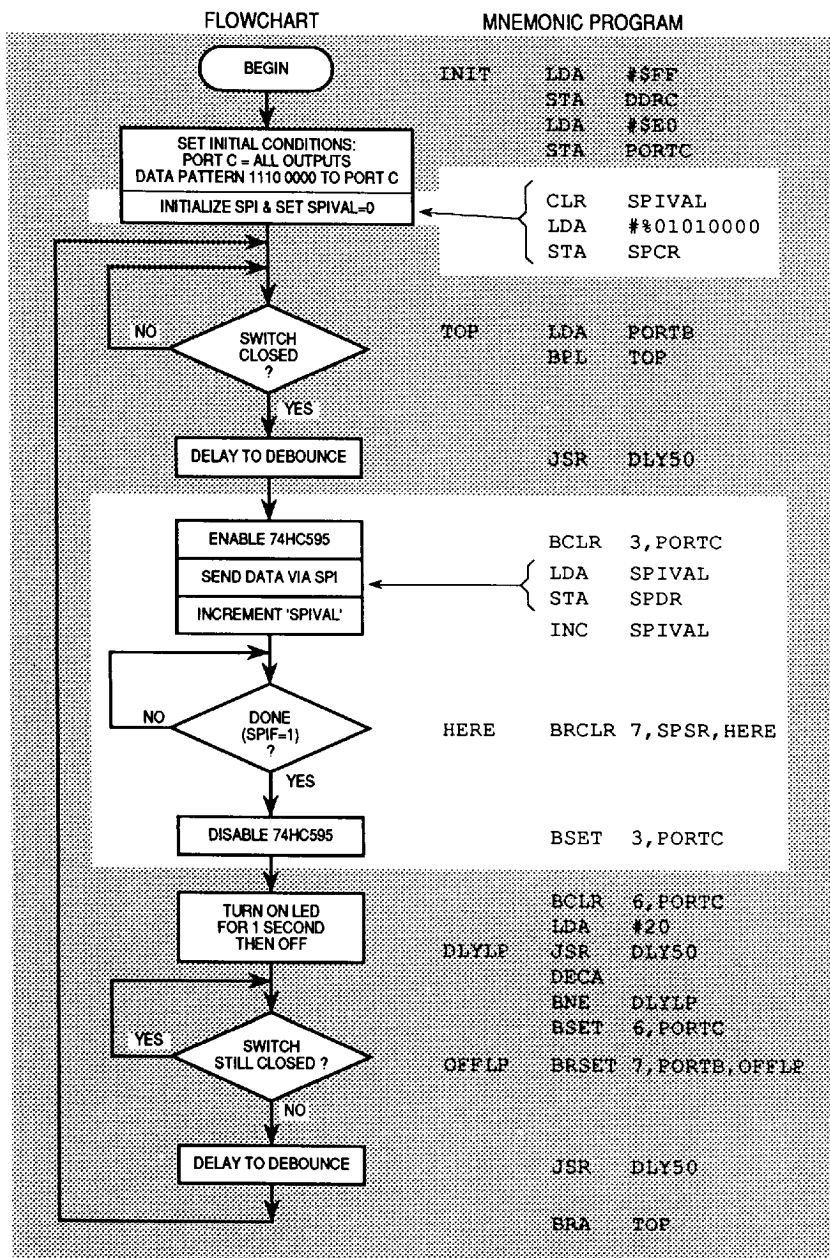
■ 6367248 0149863 T96 ■

```
FLOWCHART                    MNEMONIC PROGRAM

        BEGIN                INIT    LDA    #$FF
                                     STA    DDRC
                                     LDA    #$E0
  SET INITIAL CONDITIONS:            STA    PORTC
     PORT C = ALL OUTPUTS
 DATA PATTERN 1110 0000 TO PORT C
                                     CLR    SPIVAL
  INITIALIZE SPI & SET SPIVAL=0      LDA    #%01010000
                                     STA    SPCR


 NO      SWITCH              TOP     LDA    PORTB
         CLOSED                      BPL    TOP
         ?
        YES

  DELAY TO DEBOUNCE                  JSR    DLY50


  ENABLE 74HC595                     BCLR   3,PORTC
                                     LDA    SPIVAL
  SEND DATA VIA SPI                  STA    SPDR
  INCREMENT 'SPIVAL'                 INC    SPIVAL


 NO      DONE
         (SPIF=1)            HERE    BRCLR  7,SPSR,HERE
         ?
        YES

  DISABLE 74HC595                    BSET   3,PORTC


  TURN ON LED                        BCLR   6,PORTC
  FOR 1 SECOND                       LDA    #20
  THEN OFF                   DLYLP   JSR    DLY50
                                     DECA
                                     BNE    DLYLP
                                     BSET   6,PORTC
 YES     SWITCH              OFFLP   BRSET  7,PORTB,OFFLP
         STILL CLOSED ?
        NO

  DELAY TO DEBOUNCE                  JSR    DLY50

                                     BRA    TOP
```

NOTE: Shaded parts of this figure are identical to Figure 2-6. Unshaded instructions were added to demonstrate the SPI system.

Figure 3-35. SPI Application Example Flowchart

```
***********************************************
* Simple 68HC05 SPI Program Example        *
***********************************************

0001              PORTB  EQU   $01      Direct address of port B (sw)
0002              PORTC  EQU   $02      Direct address of port C (LED)
0005              DDRB   EQU   $05      Data direction control, port B
0006              DDRC   EQU   $06      Data direction control, port C
000a              SPCR   EQU   $0A      SPIE,SPE,-,MSTR;CPOL,CPHA,SPR1,SPR0
000b              SPSR   EQU   $0B      SPIF,WCOL,-,MODF;-,-,-,-
000c              SPDR   EQU   $0C      SPI Data Register

009e              SPIVAL EQU   $9E      One byte RAM storage location
009f              TEMP1  EQU   $9F      One byte temp storage location

0250                     ORG   $250     Program will start at $0250

0250 a6 ff   INIT    LDA   #$FF     Begin initialization
0252 b7 06           STA   DDRC     Set port C to act as outputs
             * Port B is configured as inputs by default from reset.
0254 a6 e8           LDA   #$E8     Red & grn LED & beep off, SPI EN off
0256 b7 02           STA   PORTC    Turn off red LED
             * Some pins of port C (my board) happen to be connected
             * to devices which don't apply to this example program.
             * The $E8 pattern turns off my stuff & turns off red LED

0258 3f 9e           CLR   SPIVAL   Start with 0
025a a6 50           LDA   #%01010000  SPE, MSTR, norm lo fast clk
025c b7 0a           STA   SPCR     Initialize SPI control reg

025e b6 01   TOP     LDA   PORTB    Read sw at MSB of Port B
0260 2a fc           BPL   TOP      Loop till MSB=1 (Neg trick)
0262 cd 02 86        JSR   DLY50    Delay about 50 mS to debounce

0265 17 02           BCLR  3,PORTC  Drive select of 74HC595 low
0267 b6 9e           LDA   SPIVAL   Current data to send to SPI
0269 b7 0c           STA   SPDR     Send SPI data
026b 3c 9e           INC   SPIVAL   Add one to current SPI value
026d 0f 0b fd HERE   BRCLR 7,SPSR,HERE   Wait for SPIF to set
0270 16 02           BSET  3,PORTC  Drive select of 74HC595 hi

0272 1d 02           BCLR  6,PORTC  Turn on LED (bit-6 to zero)
0274 a6 14           LDA   #20      Decimal 20 assembles to $14
0276 cd 02 86 DLYLP  JSR   DLY50    Delay 50 mS
0279 4a              DECA           Loop counter for 20 loops
027a 26 fa           BNE   DLYLP    20 times (20-19,19-18,.1-0)
027c 1c 02           BSET  6,PORTC  Turn LED back off
027e 0e 01 fd OFFLP  BRSET 7,PORTB,OFFLP  Loop here till sw off
0281 cd 02 86        JSR   DLY50    Debounce release
0284 20 d8           BRA   TOP      Look for next sw closure
```

Figure 3-36. SPI Application Example Program

■ 6367248 0149865 869 ■

overflow function allows software to extend its timing capability beyond the range of 16 bits. All activities of the timer are referenced to this one free-running counter so all timer functions have a known relationship to each other. From the MCU viewpoint, physical time is represented by the count in this free-running counter and the counter can be read at any time "to tell what time it is."

The input-capture function can be used to automatically record (latch) the time when a selected transition was detected. The output-compare function can be used to generate output signals or for timing program delays.

3.8.1 Functional Description

The timer features are as follows:
- 16-Bit Free-Running Counter with Prescaler
- Overflow Flag to Extend Timing Range
- 16-Bit Output-Compare Register
- 16-Bit Input-Capture Register
- Three Interrupt Sources

The block diagram of the timer is shown in Figure 3-37.

The programmable timer capabilities are provided by using ten addressable 8-bit registers and two external pins, output level (TCMP) and edge input (TCAP). The 10 registers are as follows:
Counter High Register, location $18
Counter Low Register, location $19
Alternate Counter High Register, location $1A
Alternate Counter Low Register, location $1B
Input-Capture High Register, location $14
Input-Capture Low Register, location $15
Output-Compare High Register, location $16
Output-Compare Low Register, location $17
Timer Control Register (TCR), location $12
Timer Status Register (TSR), location $13

Because the timer has a 16-bit architecture, the counter and alternate counter, input-capture, and output-compare values are represented by two 8-bit registers. The two 8-bit registers contain the high and low byte of each timer function value (see Figure 3-38).

■ 6367248 0149866 7T5 ■
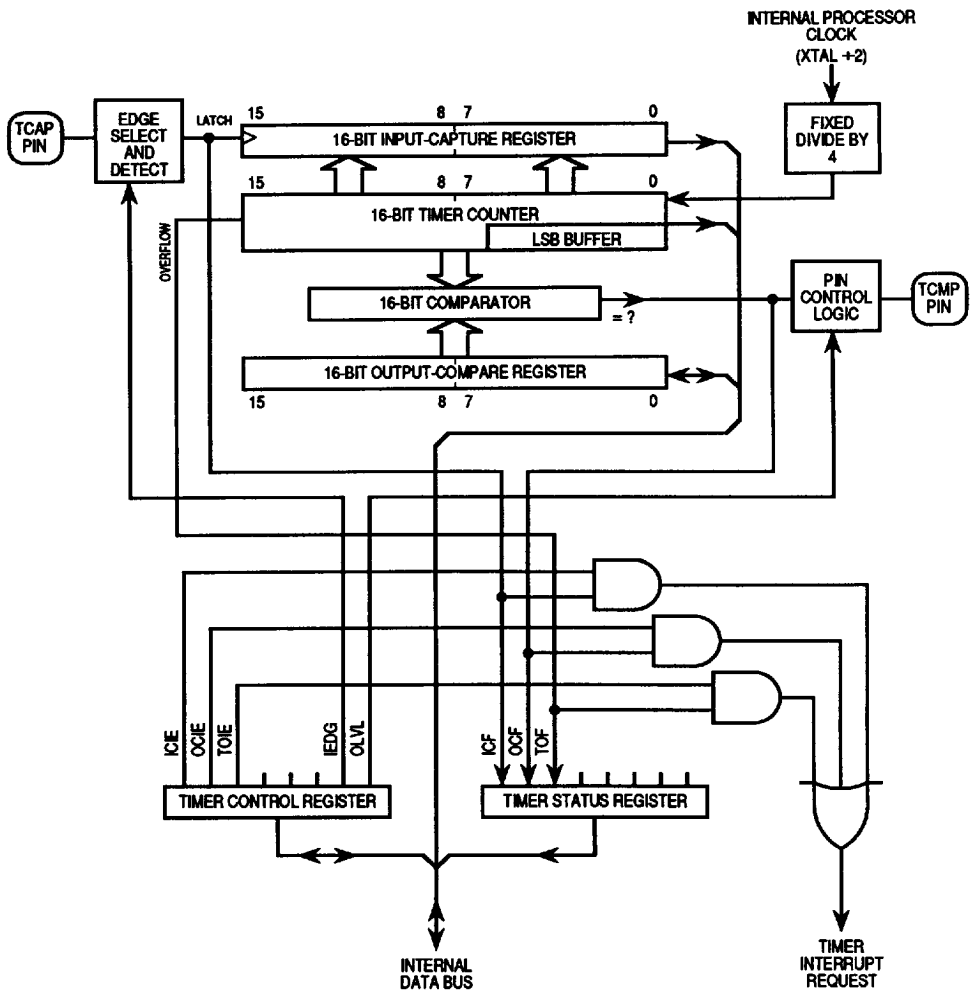
Figure 3-37. Programmable Timer Block Diagram

■ 6367248 0149867 631 ■
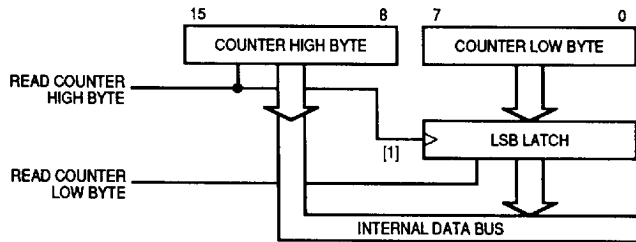
[1] - LSB latch is normally transparent, becomes latched when high byte of counter is read, and becomes transparent again when low byte of counter is read.

Figure 3-38. 16-Bit Counter Reads

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. A read from the MSB causes the LSB to be latched at the next sequential address.

NOTE

Set the I bit in the condition code register while manipulating both the high- and low-byte register of a specific timer function. This prevents interrupts from occurring between the time that the high and low bytes are accessed.

A description of each register and the external pins is given in the following paragraphs.

3.8.2 Timer Counter and Alternate Counter Registers

The 16-bit free-running counter or counter register starts from a count of $0000 as the MCU is coming out of reset and then counts up continuously. When the maximum count is reached ($FFFF), the counter rolls over to a count of $0000, sets an overflow flag, and continues to count up. As long as the MCU is running in a normal operating mode, there is no way to reset, change, or interrupt the counting of this counter. This counter, which may be read at any time to "tell what time it is," is always a read-only register.

The prescaler gives the timer a resolution of 2.0 μs if the MCU crystal is 4 MHz (internal processor clock is 2.0 MHz). Including "0", the counter repeats

every 65,536 counts ($FFFF = 65,535). Because the free-running counter is preceded by a fixed divide-by-four prescaler, the value in the free-running counter repeats every 262,144 internal processor clock cycles.

The double-byte free-running counter can be read from either of two locations $18–$19 or $1A–$1B. These registers are called the counter register and the counter alternate register, respectively.

NOTE

Normally, a timer read is made from the counter alternate register unless the read sequence is intended to clear the timer overflow flag.

If a read of the free-running counter register first addresses the most significant byte ($18), it causes the least significant byte ($19) 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 register least significant byte ($19), thus completing a read sequence of the total 16-bit counter value. The same read sequence applies to the counter alternate register. A read sequence containing only a read of the least significant byte of the free-running counter ($19) will receive the count value at the time of the read.

NOTE

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 to complete the sequence.

3.8.3 Input-Capture Concept

The input-capture function is a fundamental element of the MC68HC705C8 timer architecture. Input-capture functions are used to record the time at which some external event occurred. This is accomplished by latching the contents of the free-running counter when a selected edge is detected at the related timer input pin (edge input-TCAP pin). The time at which the event occurred is saved in the input capture register (16-bit latch). Although it may take an undetermined variable amount of time to respond to the event, software can tell exactly when the event occurred.

By recording the times for successive edges on an incoming signal, software can determine the period and/or pulse width of the signal. To measure a period, two successive edges of the same polarity are captured. To measure

■ 6367248 0149869 404 ■

a pulse width, two alternate polarity edges are captured. For example, to measure the pulse width for a high-going pulse, capture the time at a rising edge and subtract this time from the time captured for the subsequent falling edge.

When the period or pulse width is known to be less than a full 16-bit counter overflow period, the measurement is very straightforward. The counter repeats every 65,536 timer clocks, which is equivalent to 262,144 internal processor clock cycles. For period or pulse widths that extend over the full 16-bit counter period, write software to keep track of the overflows of the 16-bit counter. Examples where measurement of a period or pulse width would be used are the period of a pendulum swing or the AC line frequency (to distinquish between 50 and 60 Hz).

Another important use for the input-capture function is to establish a time reference. In this case , an input-capture function is used in conjunction with an output-compare function. For example, suppose an application requires an output signal to be activated a certain number of clock cycles after detecting an input event (edge). The input-capture function would be used to record the time at which the edge occurred. A number corresponding to the desired delay would be added to this captured value and stored in the output-compare register. Since both input captures and output compares are referenced to the same 16-bit counter, the delay can be controlled to the resolution of the free-running counter, independent of software latencies. (An example of this use would be to fire a spark plug "n" microseconds after a timing pulse is sent from the engine flywheel.)

3.8.4 Input-Capture Operation

The input capture function includes a 16-bit latch, input edge detection logic, and interrupt generation logic. The latch captures the current value of the free-running counter when a selected edge is detected at the corresponding timer input pin. The edge detection logic includes a control bit (IEDG), which enables the user's software to select the polarity of edge(s) that will be recognized. The interrupt generation logic includes a status flag to indicate that an edge has been detected and a local interrupt enable bit to determine whether or not the corresponding input-capture function will generate a hardware interrupt request. See Figure 3-39.

The two 8-bit registers (locations $14–most significant byte and $15–least significant byte) comprising the 16-bit input-capture register are read-only and are used to latch the value of the free-running counter after a defined
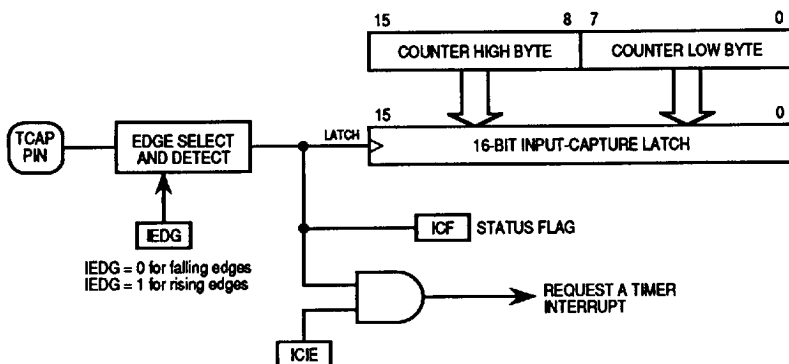
Figure 3-39. Input-Capture Operation

transition is sensed by the corresponding input-capture edge detector. The
level transition which triggers the counter transfer is defined by the input
edge bit (IEDG in the timer control register).

The free-running counter contents are transferred to the input-capture reg-
ister on each proper signal transition, regardless of whether the input-capture
flag (ICF) is set or clear. There is an uncertainty about the exact value latched
due to the resolution of the counter and synchronization delays. The input-
capture register always contains the free-running counter value, which cor-
responds to the most recent input capture. Reset does not affect the contents
of the input-capture register.

3.8.5 Output-Compare Concept

The output-compare function is also a fundamental element of the
MC68HC705C8 timer architecture. Output-compare functions are used to pro-
gram an action to occur at a specific time (i.e., when the 16-bit counter reaches
a specific value). The value in the output-compare register is compared with
the value of the free-running counter on every fourth bus cycle. When the
output-compare register matches the counter value, an output is generated,
which sets an output compare status flag and transfers the level of the OLVL
bit to the TCMP output pin (see Figure 3-40).

M68HC05 MICROCONTROLLER APPLICATIONS GUIDE MOTOROLA
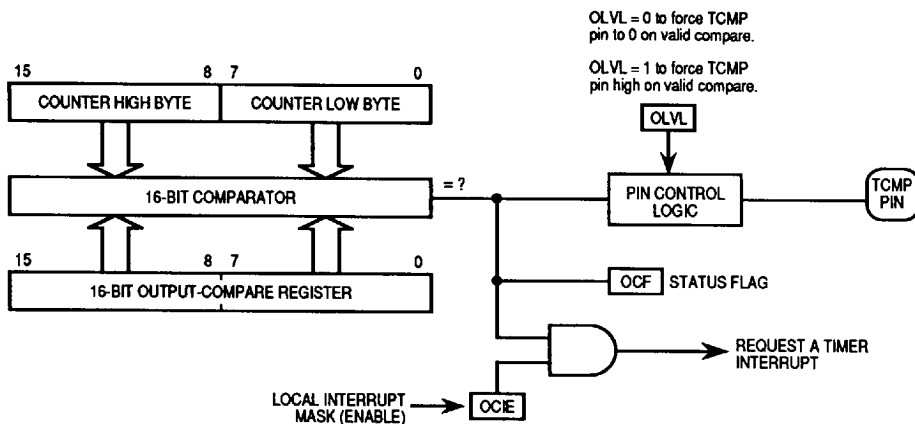
Figure 3-40. Output-Compare Operation

Change the values in the output-compare register and the output level bit after each successful comparison to control an output waveform or to establish a new elapsed timeout.

An interrupt can also accompany a successful output compare if the corresponding interrupt enable bit (OCIE) is set.

One of the easiest uses for an output-compare function is to produce a pulse of a specific duration. First, a value corresponding to the leading edge of the pulse is written to the output-compare register. The output compare is configured to automatically set the TCMP output either high or low, depending on the polarity of the pulse being produced. After this compare occurs, the output compare is reprogrammed to automatically change the output pin back to its inactive level at the next compare. A value corresponding to the width of the pulse is added to the original output-compare register value, and this result is written to the output-compare register. Since the pin-state changes occur automatically at specific values of the free-running counter, the pulse width can be controlled accurately (to the resolution of the free-running counter) independent of software latencies. By repeating the actions for generating pulses, you can generate an output signal of a specific frequency and duty cycle.

Another use of the output-compare function is to generate a specific delay. For example, suppose you want to produce a 1 millisecond delay to time

■ b3b7248 0149872 TT9 ■

programming of an EPROM byte. First, go through the initial programming steps to the point where the programming supply has been enabled (EPGM bit has been written to one). Now, read the current value of the main timer counter and add a number corresponding to 1 millisecond (XTAL = 2 MHZ, INT CLK = 1 MHz, 1 timer count = 4 μs; thus, 1 ms = 250 decimal = $FA). Write this sum to the output-compare register so that an output compare will occur when the counter gets to this value.

In this example, the actual EPROM programming time started just before the current time was read from the counter and ended after responding to the output compare and turning off EPGM. The small delays for setting up the output compare and the latency for responding to the output compare were not considered because they only make the EPROM programming time longer by a few microseconds. As you become a more advanced user of output-compare functions, you will learn how to correct these details, although it is often not necessary.

NOTE

This program would have to run from RAM since the EPROM is not usable during programming.

3.8.6 Output-Compare Operation

The output-compare register is a 16-bit register composed of two 8-bit registers at locations $16 (most significant byte) and $17 (least significant byte). 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 output-compare flag (OCF) bit is set, and the output level (OLVL) bit is clocked (by the output-compare circuit pulse) to the TCMP pin.

After a processor write cycle to the most significant byte of the output-compare register ($16), the output-compare function is inhibited until the least significant byte ($17) is also written. You must write to both bytes (locations) if the most significant byte is written first.

Because neither the output-compare flag (OCF bit) or output-compare register is affected by reset, take care when initializing the output-compare function with software. The following procedure is recommended:

1) Write to 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 clear the OCF bit if it is already set.

3) Write to the low byte of the output-compare register to enable the output-compare function.

The purpose of this procedure is to prevent the OCF bit from being set between the writes to the high and low halves of the 16-bit output-compare register. A software example follows:

```
B7    16    STA    OCMPHI    Inhibit output compare
B6    13    LDA    TSR       Clear OCF bit if set
BF    17    STX    OCMPLO    Ready for next compare
```

3.8.7 Timer Control Register (TCR)

The timer control register (see Figure 3-41) is an 8-bit read/write register containing five control bits. Three of these bits control interrupts associated with the three flag bits found in the timer status register. The other two bits control 1) which edge is significant to the input-capture edge detector (i.e., negative or positive) and 2) the next value to be clocked to the TCMP output pin in response to a successful output compare.

The TCMP pin is forced low during external reset and stays low until a valid compare changes it to a high.
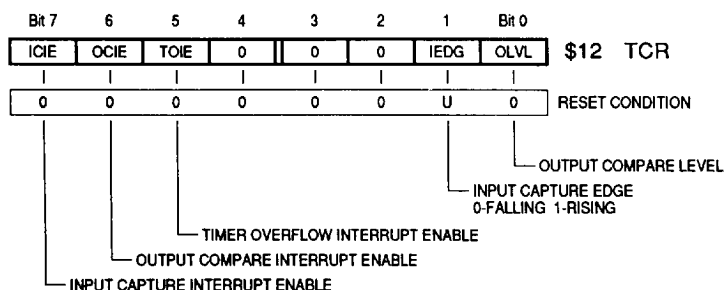
Figure 3-41. Timer Control Register

■ 6367248 0149874 871 ■

3.8.8 Timer Status Register (TSR)

The timer status register (see Figure 3-42) is an 8-bit register with three read-only bits that indicate the following status information:

1) A selected transition has occurred at the edge input (TCAP) pin 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.

3) A free-running counter transition from $FFFF to $0000 has been sensed (timer overflow).
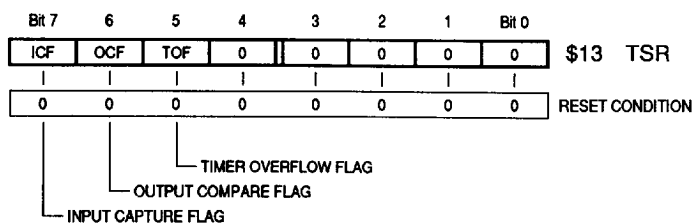
Figure 3-42. Timer Status Register

ICF
   The input-capture flag (ICF) is set when a proper edge has been sensed by the input-capture detector. It is cleared by a processor access of the timer status register (with ICF set) followed by accessing the low byte ($15) of the input-capture register.

OCF
   The output-compare flag (OCF) is set when the output-compare register contents matches the contents of the free-running counter. OCF is cleared by accessing the timer status register (with OCF set) and then accessing the low byte ($17) of the output-compare register.

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) and then accessing the least significant byte ($19) of the free-running counter.

■ 6367248 0149875 708 ■

NOTE

The counter alternate register contains the same value as the free-running counter but reading the alternate register does not clear TOF; therefore, this alternate register should be used to read the timer counter in all cases except when intending to clear TOF. This will avoid the possibility of the TOF being unintentionally cleared.

3.8.9 Timer Application Example

Figure 3-43 shows an example program to produce a 10-second delay after the timer counter is read. In this case, the timer counter and the output-compare functions are used in the software program.

The two key programming instructions that you should note are 1) the read and/or write instructions at the alternate counter and output-compare registers and 2) the addition of 16-bit numbers.

3.9 STOP/WAIT INSTRUCTION EFFECTS

The STOP and WAIT instructions put the MC68HC705C8 MCU into low power-consumption modes. These instructions also affect the programmable timer, the SCI, and the SPI systems. A STOP/WAIT flowchart is shown in Figure 3-44.

3.9.1 Low Power-Consumption Modes

The STOP instruction places the MC68HC705C8 in its lowest power-consumption mode. In the STOP mode, the internal oscillator is turned off, causing all internal processing to be halted. During the STOP mode, the I bit in the condition code register is cleared to enable external interrupts. All other registers and memory remain unaltered, and all I/O lines remain unchanged. This state continues until an external interrupt ($\overline{IRQ}$) or $\overline{RESET}$ is sensed, at which time the internal oscillator is turned on. The external interrupt or reset causes the program counter to vector to memory location $1FFA and $1FFB or $1FFE and $1FFF. These locations contain the starting address of the interrupt or reset service routine, respectively.

The WAIT instruction also places the MC68HC705C8 in a low power-consumption mode, but the WAIT mode consumes somewhat more power than the STOP mode. In the WAIT mode, all CPU processing is stopped;

```
                    **********************************************
                    * Simple 68HC05 Timer Program Example        *
                    **********************************************
0006        DDRC    EQU     $06     Data direction control, port C
0002        PORTC   EQU     $02     Direct address of port C (LED)
0016        OCMPHI  EQU     $16     Output compare high reg.
0017        OCMPLO  EQU     $17     Output compare low reg.
0013        TSR     EQU     $13     ICF,OCF,TOF,0;0,0,0,0
00a0        TENSEC  EQU     $A0     Used to count 39 out compares
00a1        TEMP    EQU     $A1     One byte temp for 16 bit OCMP add

0350                ORG     $350
0350 a6 40  INIT    LDA     #%01000000  Make DDR bit for LED a one
0352 b7 06          STA     DDRC      So Red LED pin is an output
0354 a6 40  BEGIN   LDA     #%01000000  Port C bit 6 is red LED
0356 b8 02          EOR     PORTC     Toggle red LED on PGMR board
0358 b7 02          STA     PORTC     Red LED will change every 10 Sec
035a a6 27          LDA     #39       10 sec = 38 rev + 9,632 ticks
035c b7 a0          STA     TENSEC    Counter for timer out compares

*********************************************************************
* For XTAL=2MHz (Int proc. clk=1MHz) Timer +4 makes 1 count = 4µS  *
* Counter rolls from $FFFF to 0 every 65,536 counts (262.144 mS)   *
* 10 Sec + 262.144 mS = 38 revs of timer & 9,632 counts remainder  *
* 10 Sec = 2,500,000 counts @ 4µS/count.  38 * 65,536 = 2,490,368  *
* 2,500,000 - 2,490,368 = 9632.  9632 (decimal) = $25A0            *
*                                                                  *
* To time 10 Sec, read initial count, add 9632 (remainder count)   *
* store to out compare reg ("schedule a compare").  When OCF flag =1 *
* clear it & next compare will occur when timer counts 65,536 counts *
* count the first compare plus 38 more compares (full revs).        *
*********************************************************************

035e a6 a0          LDA     #$A0    Lower half hex equiv of 9632
0360 bb 17          ADD     OCMPLO  Low half of a 16 bit add
0362 b7 a1          STA     TEMP    Temp. store until OCMPHI is added
0364 a6 25          LDA     #$25    Upper half hex equiv of 9632
0366 b9 16          ADC     OCMPHI  High half of 16 bit add (w/ carry)
0368 b7 16          STA     OCMPHI  Update OCMP hi
036a b6 a1          LDA     TEMP    Get previous saved OCMP low
036c b7 17          STA     OCMPLO  Update OCMP lo after OCMP hi

*********************************************************************
* You add low half first due to possible carry, then add high byte  *
* including any carry (ADC).  You should update out compare high    *
* byte first to avoid an erroneous compare value (compare lockout   *
* after OCMPHI till OCMPLO prevents this potential problem.          *
*********************************************************************

036e 0c 13 fd  LOOP   BRCLR 6,TSR,LOOP  Checks for out comp. flag
0371 b6 17             LDA   OCMPLO      To clear OCF flag
0373 3a a0             DEC   TENSEC      Ten seconds count down
0375 26 f7             BNE   LOOP        Loop until 10 sec done
0375 20 db             BRA   BEGIN    Repeat so PC6 toggles /10 Sec
```

Figure 3-43. Timer Application Example Program
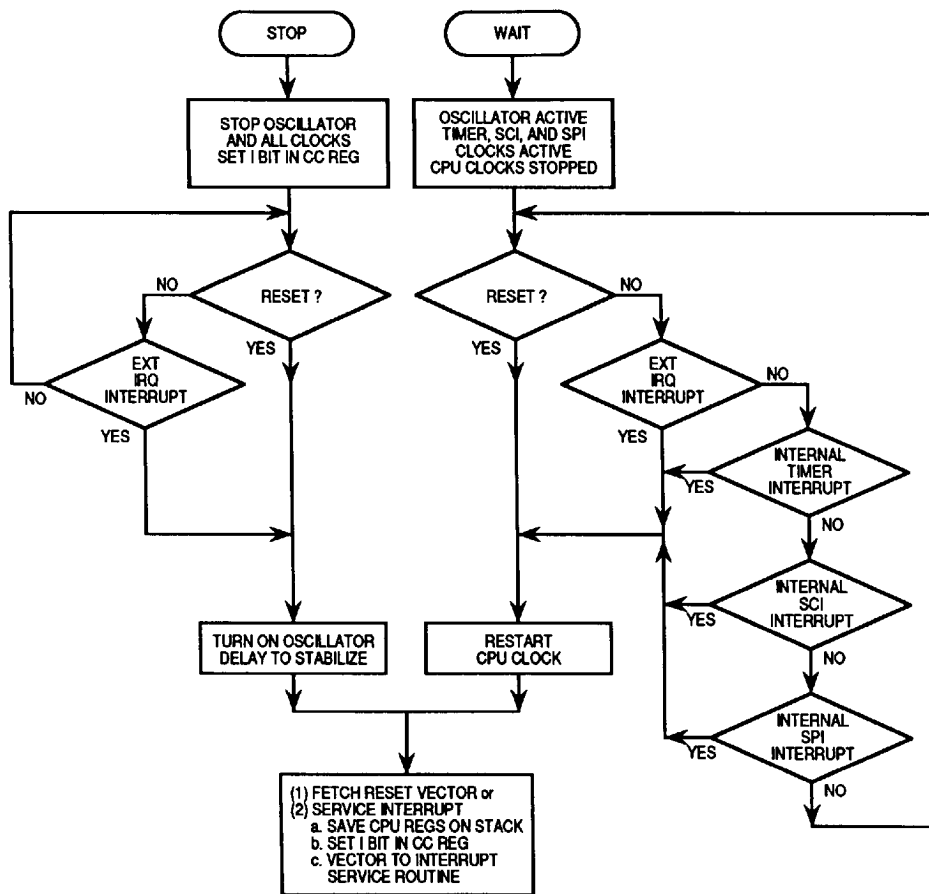
■ 6367248 0149877 580 ■

Figure 3-44. STOP/WAIT Flowchart

however, the internal clock, the programmable timer, SPI and SCI systems (if enabled) remain active. During the WAIT mode, the I bit in the condition code register is cleared to enable all interrupts. All other registers and memory remain unaltered, and all parallel I/O lines remain unchanged. This state continues until any interrupt or reset is sensed. At this time, the program counter is loaded with the interrupt vector at memory location $1FF4–$1FFF, which contains the starting address of the interrupt or reset service routine.

3.9.2 Effects on On-Chip Peripherals

The STOP instruction causes the oscillator to be turned off, which halts all internal CPU processing as well as the operation of the programmable timer, SCI, and SPI. The oscillator starts again when an external interrupt ($\overline{\text{IRQ}}$) or $\overline{\text{RESET}}$ occurs.

3.9.2.1 TIMER ACTION DURING STOP MODE. When the MCU enters the STOP mode, the timer counter stops counting (the internal processor clock is stopped). It remains at that particular count value until an interrupt or reset occurs. If the interrupt is an external low on the $\overline{\text{IRQ}}$ pin, the counter resumes from its stopped value as if nothing had happened. If a reset occurs, the counter is forced to $FFFC.

3.9.2.2 SCI ACTION DURING STOP MODE. When the MCU enters the STOP mode, the baud rate generator driving the receiver and transmitter is stopped, which halts all SCI activity.

If the STOP instruction is executed during a transmitter transfer, that transfer is halted. When the STOP mode is exited, that particular transmission resumes if the exit is the result of a low input to the $\overline{\text{IRQ}}$ pin. Since the STOP mode interferes with SCI character transmission, make sure that the SCI transmitter is idle when the STOP instruction is executed.

If the receiver is receiving data when the STOP instruction is executed, received data sampling is stopped (baud rate generator stops), and the remainder of the data is lost. The STOP mode should not be used while SCI characters are being received.

3.9.2.3 SPI ACTION DURING STOP MODE. When the MCU enters the STOP mode, the bit rate generator driving the SPI stops, halting all master mode SPI operation. Thus, the master SPI is unable to transmit or receive data. If the STOP instruction is executed during an SPI transfer, that transfer is halted until the MCU exits the STOP mode (if the exit resulted from a logic low on the $\overline{\text{IRQ}}$ pin). If the STOP mode is exited by a reset, then the appropriate control/status bits are cleared, and the SPI is disabled.

If the device is in the slave mode when the STOP instruction is executed, the slave SPI will still operate. It can still accept data and clock information in addition to transmitting its own data back to a master device. At the end of

a transmission with a slave SPI in the STOP mode, no flags are set until a logic low IRQ input results in an MCU "wake up."

When the MCU enters the STOP mode, all enabled output drivers (TDO, TCMP, MISO, MOSI, and SCK ports) remain active. Any sourcing currents from these outputs will be part of the total supply current required by the device.

3.9.2.4 WAIT MODE EFFECTS. When the MCU enters the wait mode, the CPU clock is halted. All CPU action is suspended; however, the timer, SCI, and SPI systems remain active. An interrupt from the timer, SCI, or SPI (in addition to a logic low on the IRQ or RESET pins) will cause the processor to resume normal processing.

The wait mode power consumption depends on how many systems are active. The power consumption will be greatest when all the systems (timer, TCMP, SCI, and SPI) are active. The power consumption will be least when the SCI and SPI systems are disabled (timer operation cannot be disabled in the wait mode). If a nonreset exit from the wait mode is performed (e.g., timer overflow interrupt exit), the state of the remaining systems will be unchanged. If a reset exit from the wait mode is performed, all systems revert to the (disabled) reset state.

3.10 OTPROM/EPROM PROGRAMMING

The OTPROM or EPROM programming technique is used to load a user program into the MC68HC705C8 MCU OTPROM or EPROM. This type of programming is accomplished via a bootstrap mode of operation.

3.10.1 Erasing

MC68HC705C8 EPROM MCUs are erased by exposure to a high-intensity ultraviolet (UV) light with a wavelength of 2537 angstrom. The recommended dose (UV intensity $\times$ exposure time) is 15 Ws/cm2. UV lamps should be used without shortwave filters, and the EPROM MCU should be postioned about one inch from the UV lamps.

MC68HC705C8 one-time programmable ROM (OTPROM) MCUs are shipped in an erased state and are packaged in an opaque plastic package; thus, erasing operations cannot be performed on OTPROM MCUs.

■ 6367248 0149880 075 ■

3.10.2 Programming

Programming operations are controlled by software-accessible control bits. The software program which programs the internal EPROM/OTPROM is located in either the on-chip bootstrap ROM or internal RAM.

The first programming method uses a program in the bootstrap ROM to read information from an external 8K by 8 EPROM and to program this information into the on-chip EPROM/OTPROM. The external EPROM is connected to I/O port pins of the MC68HC705C8. In this programming method, information to be programmed into the internal EPROM/OTPROM is first programmed into the external EPROM using an industry-standard PROM programmer.

A second programming method allows user programs developed on a personal computer to be downloaded to the MC68HC705C8 for programming into the on-chip EPROM/OTPROM. This method eliminates the extra steps needed to program a separate 8K by 8 EPROM. A small program that runs on the personal computer is available through the Motorola FREEWARE bulletin board service (BBS) and can be downloaded for the price of the phone call. This method is explained in Section 4 of this applications guide.

Both methods described for programming the on-chip EPROM/OTPROM ultimately use a software program running in the MCU that is being programmed. The programming software uses the program register (PROG) to control the EPROM programming process.

3.10.3 Program Register

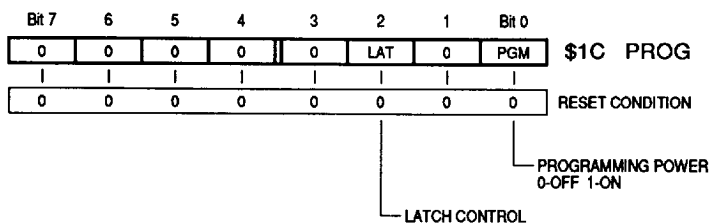The program register (see Figure 3-45) is used for PROM programming.

Figure 3-45. Program Register

■ 6367248 0149881 T01 ■

LAT

Prior to a PROM write operation, set the latch (LAT) bit. This enables the PROM data and address buses to be latched for programming a PROM location. Reset clears the LAT bit. When the LAT bit is cleared, PROM data and address buses are unlatched for normal CPU operations. This bit, which is both readable and writable, must be cleared to allow PROM read operations.

PGM

When the program (PGM) bit is set, Vpp power is applied to the PROM for programming mode of operation. Reset clears the PGM bit. This bit, which is readable, is only writable when the LAT bit is set. If the LAT bit is cleared, the PGM bit cannot be set.

3.10.4 Option Register

The option register (see Figure 3-46) is used to select memory RAM/ROM configurations, enable PROM security, and select the MCU $\overline{IRQ}$ pin sensitivity.
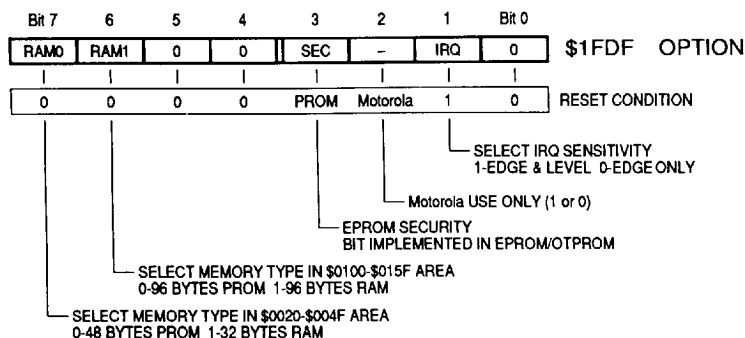

Figure 3-46. Option Register

RAM0

The RAM0 bit determines the amount and type of memory in the $0020–$005F area.

0 = 48 bytes of PROM ($0020–$005F)
1 = 32 bytes of RAM ($0030–$005F)

When RAM is selected by RAM0 = 1, the 16 bytes from $0020–$002F are unused. This bit is readable and writable at all times, allowing selection of

■ 6367248 0149882 948 ■

This Material Copyrighted By Its Respective Manufacturer

the desired memory configuration during program execution. Reset clears the RAM0 bit.

RAM1

The RAM1 bit determines the type of memory in the $0100–$015F area.
    0 = 96 bytes of PROM
    1 = 96 bytes of RAM
This bit is readable and writable at all times, allowing selection of the desired memory configuration during program execution. Reset clears the RAM1 bit.

SEC

The SEC bit is implemented as a PROM bit. During PROM programming, the SEC bit is set to enable the security feature (to disable the bootloader). This bit is normally cleared (security disabled) for an OTPROM device. For an EPROM device, clearing is accomplished by exposing the EPROM to UV light until the SEC bit is erased.

Bit 2

Factory use (logic one or logic zero).

IRQ

When the IRQ bit is set (logic one), the $\overline{\text{IRQ}}$ pin is negative edge and level sensitive. When the IRQ bit is cleared (logic zero), the $\overline{\text{IRQ}}$ pin is negative edge sensitive. Reset sets the IRQ bit. The IRQ bit can only be written once following each reset.

■ 6367248 0149883 884 ■