COPS™

COP800 Basic Family User's Manual

Customer Order Number COP8-820FAM-MAN
Publication Number 420410703-001D
February 1995

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGE
A	04/87	First Release. COPS™ COP820C/COP840C User's Manual NSC Publication Number 420410703-001.
B	10/88	Modified the program store memory section, corrected the hex and binary information for the load B pointer and the return from subroutine, and modified the description of subtract with carry. Incorporated new documentation standards where applicable.
B1	03/89	Name change. MOLE changed to microcontroller development system within text.
C	05/92	Reformatted and updated entire manual.
D	02/95	Updated appendices A & B.

PREFACE

The COP800 family of 8-bit microcontrollers is ideally suited to embedded control applications such as keyboard interfaces, electronic cordless telephones, home applications, and ABS systems. The design of this family takes advantage of National Semiconductor's M^{2}CMOS™ manufacturing technology, providing a useful combination of high performance, low power consumption, and reasonable cost. The rich instruction set and flexible addressing modes of the COP800 controllers contribute to their high performance and code efficiency.

This manual describes the features, architecture, instruction set, and usage of the COP800 microcontrollers. The first eight chapters describe the general features found in all family members. Later chapters describe the individual family members and their specific features. The following specific devices are covered:

- COP820/840/880
- COP8620/8640
- COP820CJ
- COP8780

Chapter 1, OVERVIEW, provides a general overview of COP800 family with specific feature comparisons.

Chapter 2, ARCHITECTURE, describes the overall architecture of the COP800 microcontroller, including the CPU core, registers, memory organization, reset operation, and clock options.

Chapter 3, INTERRUPTS, describes the device interrupts.

Chapter 4, TIMER, describes the on-chip timer and its various operating modes.

Chapter 5, MICROWIRE/PLUS, describes the microcontroller's MICROWIRE/PLUS serial interface and its operating modes.

Chapter 6, POWER SAVE MODE, describes an operating mode in which the microcontroller is halted, reducing power consumption almost to zero while maintaining the processor status and all register contents.

Chapter 7, INPUT/OUTPUT, describes the input/output ports of the microcontroller and how they are used.

Chapter 8, INSTRUCTION SET, describes the instruction set of the COP800 microcontrollers, including detailed descriptions of each instruction.

COPS, MICROWIRE/PLUS and M^{2}CMOS are trademarks of National Semiconductor Corporation.
TRI-STATE is a registered trademark of National Semiconductor Corporation.

Chapters 9, 10, 11, and 12 describe the specific features of the COP820/840/880, COP8620/8640, COP820CJ, and COP8780 family members, respectively.

Chapter 13, APPLICATION HINTS, contains COP800 application information.

The Appendices cover hardware development systems, emulation devices, and device electrical characterization data.

Additional information on individual COP800 family members is available from their respective data sheets.

The information contained in this manual is for reference only and is subject to change without notice.

Chapter 1 OVERVIEW

Chapter 2 ARCHITECTURE

Chapter 3 INTERRUPTS

Chapter 4 TIMER

Chapter 13 APPLICATION HINTS

Appendix A COP8 DEVELOPMENT SYSTEM

Appendix B ELECTRICAL CHARACTERIZATION DATA

INDEX

Figures

Tables

Chapter 1

OVERVIEW

1.1 INTRODUCTION

The COP800 Basic Family microcontrollers provide high-performance, low-cost solutions for embedded control applications. The 8-bit single-chip core architecture fabricated with National Semiconductor's M^2CMOS™ technology has low current drain, low heat dissipation and a wide operating voltage range. An instruction execution time of one microsecond for the majority of single-byte instructions allows high throughput; over 70 percent of the COP800 instructions are single-byte. Multiple addressing modes and a rich instruction set further enhance throughput efficiency and reduce program size. Reconfigurable I/Os, and on-chip peripherals such as multi-mode general purpose timers and the MICROWIRE/PLUS™ serial interface of the COP800 offer the flexibility needed to construct single-chip solutions for a variety of applications.

All COP800 Basic Family microcontrollers share the set of features listed in Section 1.2. Some individual family members also contain additional features. These device specific features which include special timers, brown out protection, and multi-input wakeup are listed in Section 1.3.

1.2 BASIC FEATURES

Each member of the COP800 family of microcontrollers offers the following features:

- 8-bit core processor.
- CMOS technology which provides low power, fully static operation.
- HALT MODE with very low standby power.
- Memory Mapped Architecture — all RAM, I/O Ports, and registers (except A and PC) are mapped into Data Memory Address space.
- Flexible, reconfigurable I/O.
- On-chip Data and Program Store memory.
- MICROWIRE/PLUS (3-Wire Serial Data Communications System) — allows the microcontroller to be programmed for either master or slave configuration.
- Extremely versatile 16-bit timer, with an associated 16-bit autoload/capture register, which can operate in any of 3 different modes:

- — PWM (Pulse Width Modulation).
- — External Event Counter.
- — Input Capture, with each capture resulting from an external edge input (programmable edge polarity).

- 16-bit timer and associated 16-bit autoload/capture register memory mapped as two 8-bit registers.
- Two memory mapped Control Registers for Timer Mode Select and Control, MICROWIRE/PLUS Select and Control, Interrupt Enable and Control, and Carry and Half Carry flags.
- Three interrupts:
  - — External Interrupt (maskable).
  - — Timer Interrupt (maskable).
  - — Software Trap Error Interrupt (non-maskable).
- Two 8-bit Register Indirect Data Memory Pointers
- 8-bit Stack Pointer (stack in Data Memory RAM)
- Port G. The bidirectional Port G has dual functions defined for most of the pins. The dual functions include HALT, MICROWIRE/PLUS interface, external interrupt input, timer I/O, and oscillator output.
- Three different clock modes:
  - — Crystal Oscillator
  - — R/C Oscillator
  - — External Oscillator

1.3 DEVICE SPECIFIC FEATURES

In addition to the core features, non-core features are provided by specific COP800 devices. These features are:

- Data Memory EEPROM (COP8620/8640)
- Watchdog Timer (COP820CJ)
- Brown-out Protection (COP820CJ)
- Comparator (COP820CJ)
- Modulator/Timer for High Speed PWM (COP820CJ)
- Multi-input Wakeup from HALT mode (COP820CJ)

Table 1-1 lists the available COP800 device types and shows the features present in each device. The device types are listed along the left side, and the features are listed across the top. Inside the table, the word "YES" or a numerical quantity indicates the presence of a feature; a dash indicates the absence of a feature. Memory sizes are expressed in bytes.

Table 1-1 Features List

Device Type	Program Memory		Data Memory		Timers	MIWU	Comparator	Brown Out Detection
	ROM	EPROM or OTP	RAM	EEPROM				
COP820	1K	—	64	—	1	—	—	—
COP840	2K	—	128	—	1	—	—	—
COP880	4K	—	128	—	1	—	—	—
COP8780	--	4K	128	—	1	—	—	—
COP8620	1K	—	64	64	1	—	—	—
COP8640	2K	—	64	64	1	—	—	—
COP820CJ	1K	—	64	—	3	YES	YES	YES

Chapter 2

ARCHITECTURE

2.1 INTRODUCTION

The COP800 microcontroller contains all program and data memory internally. In addition, it contains on-chip configurable I/Os, an on-chip timer and a built-in MICROWIRE/PLUS interface. The presence of on-chip memory and peripherals allows the COP800 microcontroller to provide a single-chip solution for many applications.

The COP800 memory organization is based on the "Harvard" architecture, in which the program memory is distinct from the data memory. Each of these two types of memory has its own physical memory space, and uses its own internal address bus. The advantage of this type of organization is that accesses to program memory and data memory can take place concurrently, reducing overall execution time. By contrast, in the "Von Neumann" architecture, program memory and data memory share the same address bus, and concurrent accesses cannot occur.

Except for the Accumulator (A) and Program Counter (PC), all registers, I/O ports, and RAM are memory mapped in the data memory address space. Among these registers are the B Register, X Register, Stack Pointer (SP), and I/O port registers. All such registers can be accessed by reading or writing their memory addresses.

The COP800 architecture provides one enhancement to the Harvard architecture: An instruction called Load Accumulator Indirect (LAID), which allows access to data tables stored in program memory. A conventional Harvard architecture does not allow this.

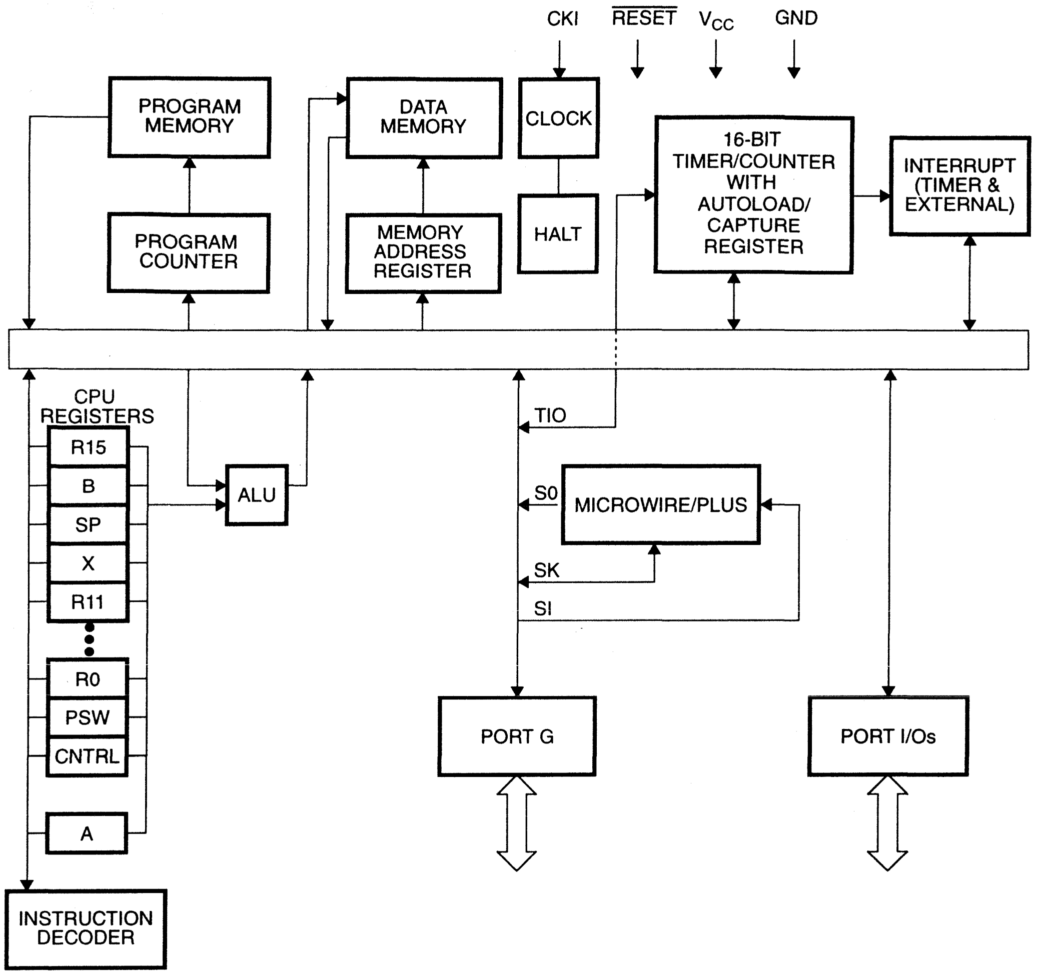
The COP800 device communicates with other devices through several configurable I/O ports or through the MICROWIRE/PLUS serial I/O interface. The I/O ports are designated by letter names such as: Port C, Port D, Port G, Port I, and Port L.

A 16-bit general-purpose timer is provided, together with an associated 16-bit autoload/capture register. The timer can be configured to operate in any of three modes: Pulse Width Modulation (PWM), external event counter, or input capture mode.

Three different interrupts are available in the device: the maskable external interrupt, the maskable timer interrupt, and the non-maskable software trap interrupt. All interrupts cause a branch to a specific address in program memory. The program code at that address determines the relative priority of the maskable interrupts.

2.2 BLOCK DIAGRAM

A block diagram of the COP800 Basic Family architecture is shown in Figure 2-1. All Basic Family devices contain the elements pictured in the block diagram. These elements include: the Arithmetic Logic Unit (ALU), Data Memory, Program Memory, Timer 1, MICROWIRE/PLUS, Port I/Os, and Interrupt Logic. Functional blocks not common to all



TSP-COP820-01

Figure 2-1 COP800 Block Diagram

Basic Family members are not shown in Figure 2-1. Block diagrams of individual devices are shown in the device specific chapters of this manual.

2.3 MEMORY ORGANIZATION

The COP800 microcontrollers are based on a modified Harvard-style architecture. This type of architecture separates the program memory from the data memory. Each memory type has its own addressing space, address bus, and data bus. The following sections describe the COP800 memory structure.

2.3.1 Program Memory

The COP800 program memory is a block of byte wide non-volatile ROM or EPROM memory, which may hold program instructions or constant data. The program memory addressing range is 32 Kbytes. A 15-bit Program Counter (PC) is used to address the program memory, which is subdivided into 4-Kbyte segments with respect to certain instructions.

The 4-Kbyte segment divisions within the program memory are related to the 2-byte Jump Absolute (JMP) and Jump Subroutine (JSR) instructions. These economical instructions cause the lower 12 bits of the PC to be replaced by the value specified in the instruction while the upper 3-bits remain unchanged. Thus, these instructions branch only within the currently addressed 4-Kbyte program memory segment.

The indirect instructions, Jump Indirect (JID) and Load Accumulator Indirect (LAID), operate only within a program memory block of 256 bytes. This restriction exists because only the lower 8 bits of the PC (PCL) are replaced during program memory table lookups. The upper 7 bits of the PC (PCU) remain unchanged. Replacing only the PCL minimizes the execution time of this instruction. Programmers must ensure that LAID and JID instructions, and their associated tables do not cross the 256 byte program memory boundaries.

The very economical Jump Relative Short (JP) instruction is completely independent of all program memory block and memory segment boundaries. This single-byte JP instruction allows a branch forward of up to 32 locations or backwards of up to 31 locations relative to the current contents of the program counter. A branch forward of 1 is not allowed, since this may be implemented with a NOP.

2.3.2 Data Memory

The COP800 data memory consists of several blocks of byte-wide RAM and/or EEPROM memory. The data memory addressing range is potentially 32 Kbytes. Devices that contain more than 128 bytes of RAM use a data segment extension register to increase the data addressing range beyond the first 128 bytes. This is necessary because the memory address register (MAR) used to access all data memory locations is only 8 bits wide.

The COP800 data memory base segment may be viewed as two separate sections: a lower address range of 0000 to 006F Hex and an upper address range of 0080 to 00FF Hex. The lower base segment contains the program stack and general-purpose data memory. The upper address range contains data registers, and the memory-mapped I/O registers, control registers, timers with associated capture registers, MICROWIRE/PLUS shift register, etc.

The data memory is either addressed directly by instructions or indirectly by the B, X and SP pointers. The COP800 instruction set permits any bit in data memory to be set, reset or tested. All I/O, registers, pointers, and counters in the COP800 family (except for A and PC) are memory mapped in data memory. Therefore, all I/O bits and register bits can be individually set, reset, and tested.

Sixteen bytes of RAM are memory mapped as "registers" at addresses 00F0 to 00FF Hex. Certain instructions work only with this register memory, while others are more efficient when used with this register memory rather than other memory. The three pointer registers X, SP and B, are memory mapped into the register memory space at address locations 00FC to 00FE Hex, respectively. In COP800 devices with more than 128 bytes of RAM, the data segment extension register is memory mapped at location 00FF Hex. See Section 2.4.4 for more information on the COP800 data registers.

The first sixteen locations of data store memory (0000 to 000F Hex) have special significance for the load B with immediate data instruction. This instruction is extremely efficient for loading the B pointer with addresses in this range because it is a single-byte, single-cycle instruction. Loading B with addresses and/or values greater than 000F Hex requires a two-byte, three-cycle instruction.

All RAM, EEPROM, I/O ports, counter, and registers (except A and PC) are mapped into the data memory address space. Table 2-1 shows a basic memory map for all COP800 devices. Refer to the device specific chapters for complete memory maps of individual devices.

Table 2-1 Data Memory Map

Address	Contents
00-6F	On-chip RAM Address Space
70-BF	On-chip Data Memory Address Space
C0-CF	I/O and Register Address Space
D0	Port L Data Register
D1	Port L Configuration Register
D2	Port L Input Pins (read only)
D3	Reserved for Port L
D4	Port G Data Register
D5	Port G Configuration Register
D6	Port G Input Pins (read only)
D7	Reserved for Port G
D8	Port I Input Pins (read only)
D9	Port C Data Register
DA	Port C Configuration Register
DB	Port C Input Pins (read only)
DC	Reserved for Port C
DE	Port D Data Register
DF	Reserved for Port D
E0-E8	On-chip Functions and Registers
E9	MICROWIRE shift register
EA	Timer 1 Lower Byte
EB	Timer 1 Upper Byte
EC	Timer 1 Autoload Register Lower Byte
ED	Timer 1 Autoload Register Upper Byte
EE	CNTRL Control Register
EF	PSW Register
F0 to FF	On-chip RAM mapped as Registers
FC	X Register
FD	SP Register
FE	B Register
FF	S Register or General Purpose Register

2.3.3 Memory Mapped I/O Registers

The COP800 devices have three different types of ports: reconfigurable input/output, dedicated output, and dedicated input. Each I/O port has specific memory-mapped I/O registers/addresses associated with it, depending on the port type. The following sections describe the I/O port register structure for each port type.

NOTE: All port registers and pins are memory-mapped in the data store memory address space. Therefore, instructions which operate on data memory also operate on port registers and pins. This includes instructions used to set, reset and test individual bits. The I/O register addresses for specific ports are listed in the memory map shown in Table 2-1.

Reconfigurable Input/Outputs

Reconfigurable input/output ports have two associated port registers: a port configuration register and a port data register. These two memory-mapped registers allow the port pins to be individually configured as either inputs or outputs, and to be individually changed back and forth in software. The configuration register is used to set up the pins as inputs or outputs. A pin may be configured as an input by writing a '0' or as an output by writing a "1" to its associated configuration register bit. If a pin is configured as an output, the associated data register bit represents the state of the pin (1 = logic high, 0 = logic low). If the pin is configured as an input, the associated data register bit determines whether the pin is a weak pull-up or Hi-Z input. Table 2-2 details the port configuration options. The port configuration and data registers are read/write registers.

Table 2-2 I/O Port Configuration

Configuration Bit	Data Bit	Port Pin Setup
0	0	Hi-Z input (TRI-STATE output)
0	1	Input with weak pull-up
1	0	Push-pull zero output
1	1	Push-pull one output

A third data memory address is assigned to each I/O port. Reading this memory address returns the value of the port pins regardless of how the pins are configured.

Dedicated Outputs

Dedicated output ports have one associated port register. This memory-mapped output data register is used to set the port pins to a logic high or low. A port pin may be individually set or reset by writing a one or zero to its associated data register bit. Port data registers may be read or written.

Dedicated Inputs

Dedicated input ports have no associated port registers. However, a data memory address is assigned to the port pins for reading of the port input. Port pin addresses are read-only memory locations.

2.4 CORE REGISTERS

All COP800 Basic Family microcontrollers share a common block of logic referred to as the COP800 core. This core includes the COP800 Central Processing Unit (CPU), the Timer 1 Block, and the MICROWIRE/PLUS block. The registers contained within these blocks are the core registers. The CPU registers include: a 15-bit program counter (PC), an 8-bit accumulator (ACC), a processor status word (PSW), a core control register (CNTRL), and sixteen 8-bit data memory registers. The Timer 1 registers include: one 16-bit timer and a 16-bit autoload capture register. The MICROWIRE/PLUS block has one 8-bit shift register. All core registers are memory mapped into the data memory address space except for the program counter (PC) and accumulator (ACC). The following sections describe in detail the COP800 core registers.

2.4.1 Accumulator

All COP800 family parts have a single 8-bit accumulator. The accumulator is used in all arithmetic and logical operations, such as ADD and XOR. In addition, it is used with the exchange, JID and LAID instructions. The arithmetic and logical instructions use the accumulator as both an operand and result register. A second operand register, if required, is either the instruction register (IR), which contains immediate data, or a register in data memory.

2.4.2 Program Counter

The CPU contains a 15-bit program counter used in addressing the byte-wide program memory. The PC is initialized to zero at reset and is incremented once for each byte of an instruction opcode. Jumps, jump subroutines, interrupts, and the JID instruction cause some or all of the PC bits to be replaced. Transfer-of-control instructions that replace only some of the PC bits have a limited jumping range.

2.4.3 Control Registers

The COP800 core contains two 8-bit control registers (PSW and CNTRL). The following paragraphs and tables show the bits contained in each register. The functions of these bits are described in later chapters.

PSW Register (Address 00EF Hex)

The Processor Status Word (PSW) register contains eight different flag bits. The register bits are assigned as follows:

GIE	Global interrupt enable (enables interrupts)
ENI	External interrupt enable
BUSY	MICROWIRE busy shifting flag
IPND	External interrupt pending
ENTI	Timer T1 interrupt enable
TPND	Timer T1 interrupt pending (timer underflow or capture edge)
C	Carry Flip/Flop
HC	Half-Carry Flip/Flop

Table 2-3 PSW Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	TPND	ENTI	IPND	BUSY	ENI	GIE

CNTRL Register (Address 00EE Hex)

The Timer and MICROWIRE Control (CNTRL) register contains various MICROWIRE/PLUS, External Interrupt Edge, and Timer Control flags. The MICROWIRE/PLUS flags include SL0 and SL1, which select the MICROWIRE PLUS clock division factor, and MSEL, which selects the G port signals G5 and G4 as the MICROWIRE/PLUS signals SK and SO, respectively. The External Interrupt Edge Control flag selects the External Interrupt Signal input polarity. The Timer Control flags include TRUN, which is used to start and stop the timer/counter, and three Timer Mode Control signals.

The timer and MICROWIRE control register bits are:

SL1 & SLO	Select the MICROWIRE clock divide-by (00=2,01=4,1x=8)
IEDG	External interrupt edge polarity (0 = rising edge, 1 = falling edge)
MSEL	Selects G5 and G4 as MICROWIRE signals SK and SO, respectively
TRUN	Used to start and stop the timer/counter (1 = run, 0 = stop)
TC1	Timer T1 Mode Control Bit
TC2	Timer T1 Mode Control Bit
TC3	Timer T1 Mode Control Bit

Table 2-4 CNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1	TC2	TC3	TRUN	MSEL	IEDG	SL1	SLO

2.4.4 Data Registers

The COP800 contains sixteen 8-bit data registers located in data memory from address 00F0 to 00FF Hex. Four of these registers, 00FC through 00FF Hex, have special functions. Locations 00FC and 00FE Hex contain the 8-bit data memory pointers X and B, respectively. Location 00FD contains the 8-bit stack pointer (SP) for data memory.

Location 00FF is reserved for the data segment extension register, which is used in some COP800 devices to extend data memory beyond 128 bytes. In devices that contain 128 or fewer bytes of data memory, this register is available for general usage. The remaining twelve registers, 00F0 through 00FB, are always available for general purpose use.

Certain COP800 instructions differentiate data registers from other data memory locations, such as the DRSZ (decrement register skip if zero) instruction. DRSZ subtracts one from a specified data register and skips the following instruction if the result of the decrement is zero. This instruction is extremely useful in constructing code loops, and makes the data registers ideal choices for loop counters. Other instructions like the "load memory with immediate data" are more efficient when used with the register memory than when used with the general data memory.

Stack Pointer

The stack pointer (SP) is memory mapped at data memory location 00FD Hex. The stack pointer should be initialized before any subroutine calls or interrupts occur. Normally, the stack pointer is initialized to the top of the base segment of data memory. In the COP820C, this is memory location 002F Hex. In the COP880C, this is memory location 006F Hex.

Pushing addresses onto the stack causes the stack to grow downward in data memory toward address zero. Popping addresses off the stack causes the stack to shrink upward. If the stack pointer is initialized to the top of the base segment of memory, over-popping the stack causes a Software Trap error interrupt. The lower limit of the stack is address 0000 Hex. Over-pushing the stack causes the stack to wrap around to addresses 00FF and 00FE Hex (subroutine calls and interrupts cause a double-byte push). This should be avoided because it interferes with the B pointer, which is memory mapped at location 00FE Hex.

The user may initialize the stack pointer anywhere in the base segment of memory. The stack still grows down toward address zero, but the stack no longer has the Software Trap interrupt over-pop protection. Initializing the stack pointer to one of the upper base segment data register addresses (00F0 to 00FB Hex) is potentially very hazardous. The available stack memory is severely limited, and if the stack pushes downward beyond address location 00F0, interference occurs with the PSW and CNTRL control registers, which are memory mapped at address locations 00EF and 00EE Hex, respectively.

Data Memory Pointers (Index Registers)

The COP800 contains two special registers, X and B, which may be used as pointers. These registers allow indirect addressing of all locations mapped in the data memory address space. In addition, these registers may be automatically incremented or decremented by certain instructions that use register indirect addressing. The auto-incrementing and auto-decrementing features allow the user to easily step through data memory locations (i.e., tables).

2.4.5 MICROWIRE/PLUS Register

The MICROWIRE/PLUS three-wire serial communication system contains an 8-bit memory mapped serial shift register (SIOR). The serial data input and output signals to the SIOR register are supplied by SI and SO, respectively. The shift register is clocked by signal SK. Data is shifted through the SIOR from the low-order end to the high-order end on the falling edge of the SK clock signal.

2.4.6 Timer Registers

The COP800 core contains one timer block. The timer block consists of a 16-bit timer/counter with an associated 16-bit autoload/capture register. The 16-bit register and timer are each organized as two 8-bit memory mapped registers. The upper and lower byte addresses for the memory mapped timer and autoload/capture register are shown in the data memory address map (Table 2-1).

2.5 CPU OPERATION

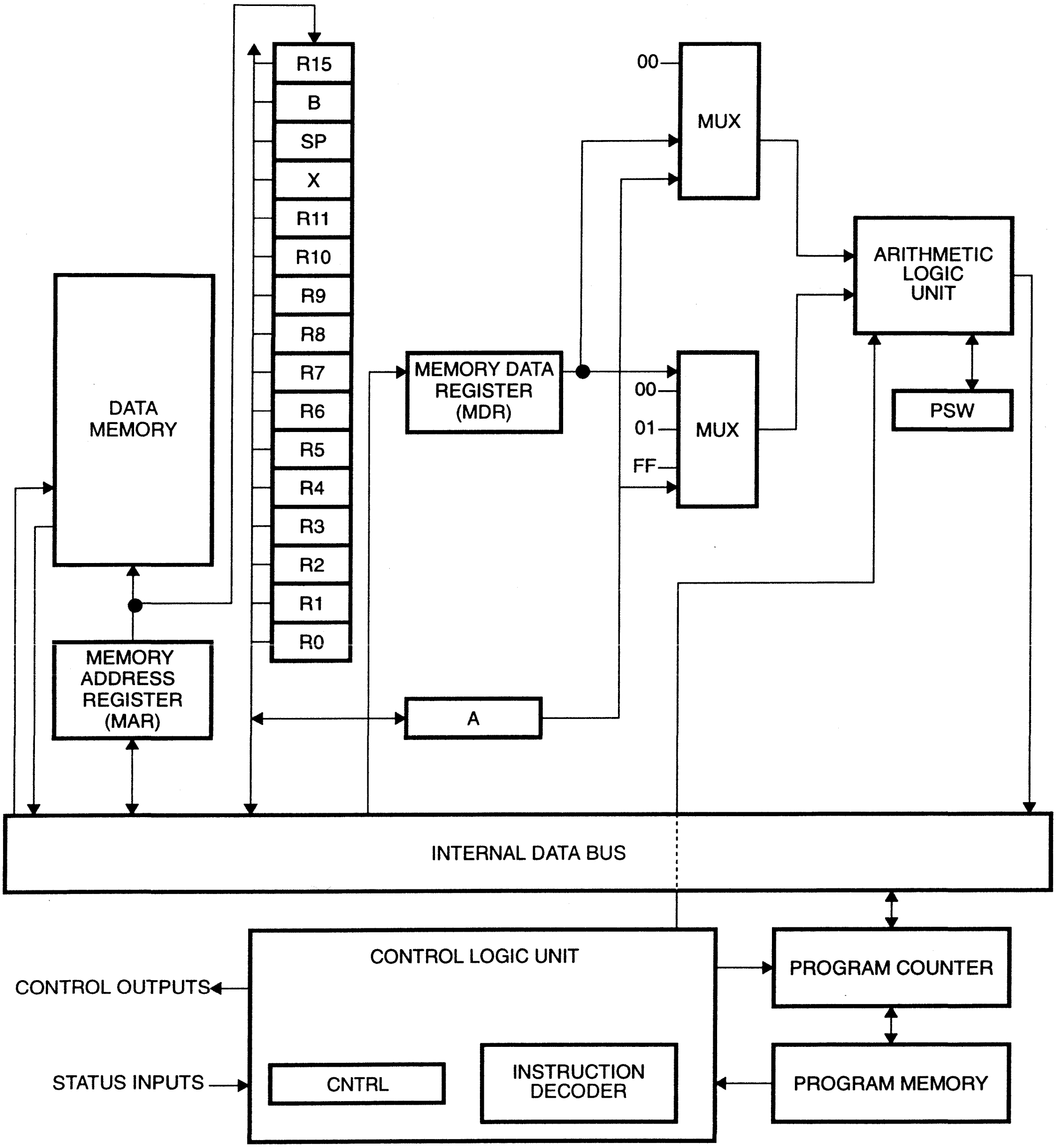
This section describes the operation of the COP800 Central Processing Unit (CPU). A brief description of the control logic and the Arithmetic Logic Unit, is given at the beginning of this section. The remainder of this section describes how the microcontroller performs memory fetches, executes instructions, and handles interrupt and error conditions. A block diagram of the main elements which interface with the control logic and ALU is shown in Figure 2-2.

Control Logic

The CPU Control Logic controls virtually all operations within the device. It includes the program counter, the memory address register, the processor status word register, and the instruction register for storing information. It also includes logic for directing memory fetches, instruction decoding and execution, and interrupt/error handling. It receives inputs from the ALU and on-chip peripherals, including the timer(s) and the MICROWIRE/PLUS interface, and generates control signals for these and other parts of the device.

Arithmetic Logic Unit (ALU)

The ALU performs all logical and arithmetic operations. Inputs to the ALU are provided by the accumulator, several hard-wired data constants, the carry/half-carry bits and the memory data register (MDR). The ALU inputs for a given instruction are specified in the instruction opcode. The accumulator functions as both a source and destination for the ALU, and is used in **all** logical and arithmetic instructions. It always contains the result of the last executed logical, arithmetic, or load/exchange accumulator instruction. The hard-wired data constants, which include 0000, 0001, and 00FF Hex, are used in instructions like CLR A, INC A, and DEC A. These instructions have an implicit addressing mode. The carry (C) and half-carry (HC) bits are used in instructions like ADC and SUBC. All arithmetic and logical instructions with two operands use the MDR as one input to the ALU. The MDR may be loaded with operands from data memory or



TSP-COP820-02

Figure 2-2 COP800 CPU Interface

the instruction register (immediate data specified in an instruction opcode). Since only one MDR exists, arithmetic and logical instructions can not be performed directly on two operands from data and/or program memory. Such operations require one operand from memory to be loaded into the accumulator prior to execution.

2.5.1 Memory Fetches

The following two sections describe the manner in which the COP800 Basic Family microcontrollers access data and program memory. Memory access time greatly affects total instruction execution time, and is therefore an important element in understanding the COP800 CPU timing.

Data Memory Fetches

All data memory accesses are performed using the internal memory address register (MAR). The contents of the MAR select the location within the data memory address space to be read/written by the current instruction. It should be noted that Memory Direct to Memory Direct data transfers and operations are not supported.

The MAR is loaded with the contents of the B pointer during the last instruction cycle of all instructions. Therefore, instructions that use the Register B Indirect mode of addressing are extremely efficient. This is because the address of the memory location to be accessed during an instruction is already present in the MAR at the start of the instruction. Instructions that use Memory Direct addressing or Register X Indirect addressing to access data memory require an extra one or two instruction cycles to fetch and load the desired memory address into the MAR before the actual instruction can be executed.

Some instructions that use Memory Direct addressing are more efficient when addressing the data registers located between 00F0 and 00FF Hex because in these instructions, the complete memory address of the register is contained in the first byte of the instruction opcode. This allows the MAR to be loaded with the new address in the first instruction cycle of the instruction. Instructions that do not access data memory do not affect the MAR. During the execution of instructions that use the ALU and an operand from data memory, the contents of the memory location addressed by the MAR is loaded into the memory data register (MDR) before being fed into the ALU.

Program Memory Fetches

All program memory accesses are performed using the 15-bit program counter (PC). This includes accesses to program memory for table lookups. At any given time, the PC addresses one byte within program memory. This byte is loaded into the instruction register for decoding, or used as immediate or memory address data. All data/opcode fetches cause the PC to be incremented automatically, so that the PC typically points to one program memory location ahead of the current instruction byte being executed. This allows pre-fetching of opcodes. This is also the reason why table lookup instructions (LAID, JID) located at the last byte within a 256-byte program memory page cause fetches from program memory locations in the following 256-byte page. (The JID and LAID instructions replace the lower 8 bits of the PC, and rely on the current upper 7 bits of the PC to form the complete address for table lookups. However, the upper 7 bits of the PC change when the PC is automatically incremented over a page boundary.)

2.5.2 Instruction Decoding and Execution

All instruction decoding is performed by the CPU Control Logic. Single-byte opcodes require a single memory fetch. Therefore, many single-byte opcodes are single cycle. Multiple byte opcodes require more than one program memory fetch. The first byte of these opcodes is decoded to determine the number of program fetches needed to complete the instruction, and possibly the actual operation to be performed. Only one program memory fetch can be performed during a single instruction cycle. Therefore, an instruction always requires at least as many instructions cycles to execute as the number of opcode bytes.

NOTE: Data and program memory fetches can be performed in the same instruction cycle due to the Harvard-style architecture of the COP800 Family.

The instruction cycle clock (t_C) always equals one-tenth the frequency of the clock signal at the CKI pin. All instructions are executed in multiples of the instruction cycle clock period.

A pre-fetch of the next instructions first byte is always done during the last cycle of an instruction. In addition, the PC is always incremented. This means that at the start of the first cycle of an instruction, the opcode for that instruction is already in the IR and the PC is pointing to the next instruction byte. In order to generate skips (non-execution of an instruction), the microcontroller Skip Logic is activated. This prevents the next instruction (already located in the IR) from being executed by the microcontroller. Skipped instructions require X number of cycles to be skipped, where X equals the number of bytes in the skipped instruction's opcode.

The exact number of instruction cycles required for an instruction to execute can be found in Section 8.6.2. As noted previously, memory fetches (and therefore addressing modes) greatly influence instruction execution time. In order to optimize instruction execution time, the user should pay special attention to these items when developing code.

The following sections detail the steps performed by the CPU when executing different instructions.

One-Cycle Instructions

During the single cycle of these instructions, the following steps are performed by the CPU:

1. The instruction is decoded and executed. (The instruction opcode is already in the IR at the start of the instruction cycle due to pre-fetching).
2. The next instruction is fetched from program memory.
3. The PC is incremented.

Two-Cycle Instructions

The COP800 two-cycle instructions have either one or two byte opcodes. They fall into one of five instruction categories: logical, arithmetic, conditional, exchange or load. The CPU steps for the various 2-cycle instructions are given below.

The logical, arithmetic and conditional instructions that use the Immediate addressing mode have the following steps:

Cycle 1: Decode the opcode for the instruction. Fetch the immediate data from program memory. Execute the instruction. Activate Skip Logic if necessary. Increment the PC. (The logical/arithmetic or conditional instruction is complete at the end of this instruction cycle.)

Cycle 2: Fetch the first byte of the next instruction. Increment the PC.

Two-cycle load and exchange accumulator instructions, and load memory indirect using the B pointer have these steps:

Cycle 1: Decode the opcode for the instruction. If necessary, fetch the immediate data from program memory and increment the PC. Execute the instruction. (The load or exchange is complete at the end of this instruction cycle.)

Cycle 2: If necessary, increment or decrement the B pointer. Load the contents of the B pointer into MAR. Fetch the first byte of the next instruction. Increment the PC.

Three Cycle Instructions

The COP800 devices have eleven three-cycle load and exchange instructions. A generic overview of the sequence of steps performed by the CPU in executing these instructions is given below.

Cycle 1: Decode the opcode for the instruction. If necessary, fetch the memory direct address from program memory and increment the PC. Load the MAR with the address of the data memory location to be accessed (either the address fetched from program memory or the contents of the X pointer, depending on the instruction). If necessary, increment or decrement the X pointer.

Cycle 2: If necessary, fetch the immediate data from program memory and increment the PC. Execute the instruction. (The load or exchange is complete at the end of this instruction cycle.)

Cycle 3: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

The remaining three-cycle instructions are all unique. Therefore, the CPU sequence of events is given separately for each.

JP Instruction

Cycle 1: At the beginning of this instruction cycle, the PC is one count ahead of the address of the JP instruction. Decode the instruction opcode. Add the lower six bits of the contents of the IR (the JP opcode) to the lower byte of the PC.

Cycle 2: If the offset contained in the JP opcode was positive and the add performed in Cycle 1 had a carry out (overflow), increment the upper byte of the PC. If the offset was negative and no carry out was produced by the add in Cycle 1 (underflow), decrement the upper byte of the PC.

Cycle 3: Fetch the first byte of the next instruction (instruction located at the branch address). Increment the PC.

JMP Instruction

Cycle 1: Decode the instruction opcode. Fetch the lower byte of the branch address from program memory. Load the lower byte of the PC with the fetched address.

Cycle 2: Load the four least significant bits of the JMP opcode stored in the IR into the four least significant bits of the upper byte of the PC.

Cycle 3: Fetch the first byte of the next instruction (instruction located at the branch address). Increment the PC.

LAID Instruction

Cycle 1: Decode the instruction opcode. Exchange the lower byte of the PC with the contents of the accumulator.

Cycle 2: Fetch the byte from program memory addressed by the PC. Transfer the contents of the accumulator back to the lower byte of the PC. Store the fetched byte in the accumulator.

Cycle 3: Fetch the first byte of the next instruction. Increment the PC.

JID Instruction

Cycle 1: Decode the instruction opcode. Exchange the lower byte of the PC with the contents of the accumulator.

Cycle 2: Fetch the byte from program memory addressed by the PC. Transfer the contents of the PC back to the accumulator (restore the contents of the ACC). Store the fetched byte in the lower byte of the PC.

Cycle 3: Fetch the first byte of the next instruction. Increment the PC.

DRSZ Instruction

Cycle 1: Decode the opcode of the instruction. Load the MAR with the address of the register being decremented.

Cycle 2: Decrement the contents of the register addressed by the MAR. If the result is zero, activate the Skip Logic.

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

Four-Cycle Instructions

All 4-cycle instructions except JMPL use the Memory Direct addressing mode. The following steps outline the general sequence of events performed by the CPU during the execution of these memory direct instructions.

Cycle 1: Decode the Memory Direct mode opcode prefix (which is already in the IR because it was fetched during the previous instruction). Fetch the memory direct address from program memory and store it in the MAR. Increment the PC.

Cycle 2: Fetch the actual opcode from program memory and store it in the IR.

Cycle 3: Execute the instruction. (The bit manipulation, conditional test, or logical/ arithmetic operation is complete at the end of this instruction cycle.)

Cycle 4: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

A JMPL has the following steps:

Cycle 1: Decode the JMPL opcode. Fetch the second byte of the instruction (the high-order byte of the branch address) and store it in IR. Increment the PC.

Cycle 2: Fetch the third byte of the instruction (the low-order byte of the branch address) and load it into the lower byte of the PC.

Cycle 3: Load the high-order byte of the branch address from the IR into the upper byte of the PC.

Cycle 4: Fetch the next instruction (located at the branch address). Increment the PC.

Five-Cycle Instructions

The COP800 devices have only five 5-cycle instructions. These instructions are JSR, JSRL, RET, RETI and RETSK. All of these instructions force program branches.

The CPU performs the following steps during the JSR and JSRL instructions:

Cycle 1: Decode the opcode for the instruction. Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location. If JSRL, fetch the next byte of the instruction and increment the PC.

Cycle 2: Increment the PC. Push the low-order byte of the return address onto the stack (store at the location addressed by MAR). Fetch the next byte of the instruction. Load the low-order byte of the subroutine address (addressed by the MAR) into the PC.

Cycle 3: Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 4: Push the high-order byte of the return address onto the stack (store at the location addressed by MAR). If JSR, load the four bits of the high-order byte of the subroutine address stored in the IR into the PC. If JSRL, load the seven bits of the high-order byte of the subroutine address stored in the IR into the PC.

Cycle 5: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

The CPU performs the following steps during the RET, RETSK, and RETI instructions:

Cycle 1: Decode the opcode for the instruction. Increment the stack pointer to point to the last entry on the stack. Load the MAR with the address of the last entry in the stack (address in the updated SP).

Cycle 2: Pop the high-byte of the return address off the stack (the contents of the memory location addressed to by the MAR). Load the upper byte of the PC with the high byte of the return address.

Cycle 3: Decode the opcode for the instruction. Increment the stack pointer to point to the last entry on the stack. Load the MAR with the address of the last entry in the stack (address in the updated SP).

Cycle 4: Pop the low byte of the return address off the stack (the contents of the memory location addressed to by the MAR). Load the lower byte of the PC with the low byte of the return address.

Cycle 5: Load the contents of the B pointer into the MAR. If RETI, set the GIE bit. If RETSK, activate skip logic to skip the instruction at the return address. Fetch the first byte of the instruction at the return address. Increment the PC.

Seven-Cycle Instructions

The Software Trap is the only instruction which requires seven cycles to execute. Refer to Section 2.5.3 for information on the execution of this instruction.

2.5.3 Interrupt and Error Handling

The COP800 Basic Family microcontrollers have three interrupt sources; External, Timer 1 and Software Trap. All interrupts cause the CPU to force a jump to location 00FF Hex in program memory. Therefore, all interrupt and error handling routines or branches should be located at 00FF Hex.

The CPU forces a jump to 00FF Hex by jamming the INTR opcode (00) into the IR upon detecting an interrupt or error. An interrupt that occurs while an instruction is being executed is not acknowledged until the end of the current instruction. If the instruction following the current instruction is to be skipped, the next instruction is skipped before the pending interrupt is acknowledged. Once an interrupt/error is acknowledged, the CPU requires seven cycles to perform the jump to location 00FF Hex. The sequence of cycles is:

Cycle 1: Jam opcode 00 into the IR. If not a Software Trap, reset the GIE bit. Decrement the lower-byte of the PC. (Note: The address of the instruction that was ready to be executed is the return address to be saved on the stack. However, the PC is one count ahead of the current instruction, and must therefore be decremented before being saved on the stack.)

Cycle 2: If the decrementing of the lower-byte of the PC caused a borrow, decrement the upper byte of the PC.

Cycle 3: Load the MAR with the address of the first available stack location (the address currently in SP). Increment the stack pointer to point to the next byte of data on the stack.

Cycle 4: Push the low-order byte of the return address onto the stack (store at the location addressed by MAR). Load the low-order byte of the PC with 0FF Hex.

Cycle 5: Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 6: Push the high-order byte of the return address onto the stack (store at the location addressed by MAR). Load the upper byte of the PC with 00 Hex.

Cycle 7: Load the contents of the B pointer into the MAR. Fetch the first byte of the instruction located at 00FF Hex. Increment the PC.

Once a branch to location 00FF Hex occurs, the user software must poll the available pending flags to determine the source of the interrupt. If no pending flags are set, the software should assume a Software Trap has occurred and should take appropriate action. Refer to the Interrupt Chapter for more information on interrupts and the Software Trap.

2.6 RESET

The COP800 enters a reset state immediately upon detecting a logic low on the $\overline{\text{RESET}}$ pin. When the $\overline{\text{RESET}}$ pin is pulled to a logic high, the device begins code execution within two instruction cycles. The $\overline{\text{RESET}}$ pin must be held low for a minimum of one instruction cycle to guarantee a valid reset. During power-up initialization, the user hardware must ensure that the $\overline{\text{RESET}}$ pin is held low until the COP800 is within the specified V_{CC} voltage. Additionally, the user must ensure that the oscillator has had time to stabilize. An R/C circuit on the $\overline{\text{RESET}}$ pin with a delay 5 times greater than the power supply rise time is recommended.

All COP800 microcontrollers contain logic to initialize their internal circuitry during the reset state. The following initializations are performed at reset:

- The Program Counter is loaded with 0000 Hex.
- All bits of the PSW and CNTRL registers are reset. This disables all interrupts, stops Timer 1, and disables MICROWIRE/PLUS.

The Accumulator and all data memory and data registers, including the B, X and SP pointers, are uninitialized at reset.

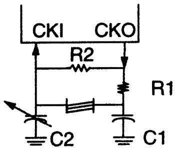
Refer to the device-specific chapters for details on the reset initialization of registers not found in the COP800 microcontroller core.

2.7 CLOCK OPTIONS

Most COP800 parts support three clock options; crystal oscillator, RC oscillator, and external oscillator. Depending on the device type, the clock option is either selected via a mask option or programmed into the device by the user. Selection of a specific clock option affects the operating frequency, clocking accuracy, and power consumption of a particular device. Refer to the device specific data sheets to obtain accurate information on frequency ranges, power consumption, and component values for the different oscillator circuits.

2.7.1 Crystal Oscillator

The dedicated CKI (clock input) pin and G7 (CKO) on the COP800 devices can be connected to make a crystal controlled oscillator as shown in Figure 2-3. If G7 is used as the CKO pin, it is not available for general purpose use.

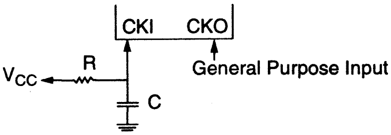


COP820-09-F

Figure 2-3 Crystal Oscillator Circuit

2.7.2 RC Oscillator

The dedicated CKI pin can be used to construct an RC oscillator as shown in Figure 2-4. With this option, G7 is available as a general purpose input pin.



COP820-10-F

Figure 2-4 RC Oscillator Circuit

2.7.3 External Oscillator

The dedicated CKI pin can be driven by an external clock signal that meets specified duty cycle, rise/fall times, and input levels. With this option, G7 is available as a general purpose input pin. See Figure 2-5.

CKI
CKO
External Clock
COP820-11-F

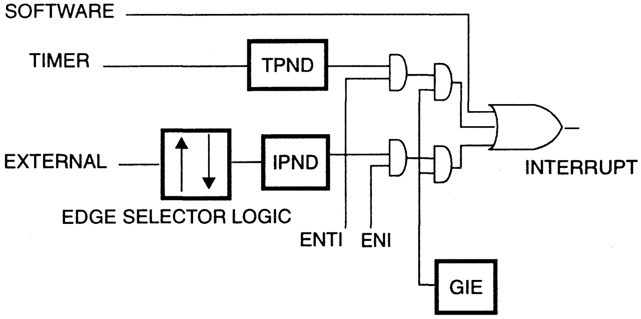
Figure 2-5 External Oscillator Circuit

Chapter 3

INTERRUPTS

3.1 INTRODUCTION

All COP800 Basic Family members have three independent interrupt sources: External, Timer 1 and Software Trap. These interrupts may be divided into two categories, maskable and non-maskable. The External and Timer 1 interrupts are both software maskable. The Software Trap is non-maskable because it is used to detect errors in program execution. The COP800 processes all interrupts similarly by halting normal program execution and forcing a branch to program memory location 00FF Hex. The user program determines how interrupts are prioritized and serviced. A block diagram of the interrupt logic found in all COP800 Basic Family members is shown in Figure 3-1.



TL/DD/10802-8

Figure 3-1 Interrupt Block Diagram

Maskable interrupts, in addition to the External and Timer 1 interrupts, are available on some devices. These interrupts are discussed in detail in the appropriate device specific-chapter near the end of this manual. General information regarding interrupt processing and maskable interrupts contained in this chapter pertains to all COP800 interrupts.

3.2 INTERRUPT PROCESSING

An interrupt is an asynchronous event which may occur before, during, or after an instruction cycle. Any interrupt which occurs during the execution of an instruction is not acknowledged until the start of the next normally executed instruction. If the next normally executed instruction is to be skipped, the skip is performed before the pending interrupt is acknowledged. All maskable interrupts set their associated interrupt pending flags immediately upon occurrence.

At the start of an interrupt acknowledgment, the CPU control logic halts normal program execution by jamming a zero opcode (00) into the instruction register. The microcontroller then performs the following actions:

1. If the interrupt is not a software trap, the Global Interrupt Enable bit is reset. This prevents other MASKABLE interrupts from being acknowledged while another interrupt is being serviced. The software trap does not reset GIE. Therefore, the software trap may be interrupted by other interrupts (causing nested interrupts).
2. The address of the next normally executed instruction is saved on the system stack. (A software trap error instruction pushes its own address onto the system stack.)
3. The Program Counter is loaded with 00FF Hex. This forces a jump to the user's general interrupt service routine, which is always stored at location 00FF Hex in program memory.

The COP800 requires seven instruction cycles to perform the actions listed above. Maskable interrupts that occur during this time still set their associated interrupt pending flags. Detailed information about the microcontroller's processing of interrupts is provided in Section 2.5.3, Interrupt and Error Handling.

The interrupt service routine at location 00FF Hex should determine the source(s) of the interrupt. This is accomplished by reading the interrupt pending flags. If more than one pending flag is set, the software must determine the relative priority of the interrupts. If no pending flags are set, the software should assume a software trap has occurred. Once the source and priority of an interrupt have been determined, the program should branch to an appropriate service routine. Since all interrupts force a branch to location 00FF Hex, all necessary context switching (saving of the accumulator, B pointer etc.) may be performed prior to branching to a specialized service routine. At the end of the service routine, the software should execute an instruction to return to normal program flow.

3.3 MASKABLE INTERRUPTS

The COP800 allows all maskable interrupts to be individually enabled and disabled in software. Each maskable interrupt has an associated enable bit which is used for this purpose. In addition, each interrupt has an associated pending flag. This pending flag is always set by hardware immediately upon the occurrence of an interrupt regardless of whether or not the associated enable bit is set. The pending flag is polled by the user to determine the source of an interrupt. Interrupt enable bits are set and reset by software. Interrupt pending flags are set by hardware but must be reset by software. Pending flags may also be set by software to force interrupts. When setting a maskable interrupt enable bit, it should always be considered whether or not a previously pending occurrence of the interrupt is to be acknowledged. If previous occurrences are to be ignored and only new occurrences acknowledged, then the associated pending bit should be reset before the enable bit is set.

All maskable interrupts which have been individually enabled must be globally enabled/disabled by setting/resetting the Global Interrupt Enable (GIE) bit located in the

Processor Status Word (PSW) register. An interrupt will only be acknowledged if its associated interrupt enable bit and the GIE bit are set.

The acknowledgment of a maskable interrupt always forces the reset of the GIE bit. This prevents subsequent maskable interrupts from interrupting a service routine already in progress. Interrupts that occur during the servicing of a previous interrupt are not lost. These interrupts set their associated interrupt pending flags, and are acknowledged after the return from the current interrupt service routine. Resetting of the GIE bit does not prevent a non-maskable software trap from interrupting the microcontroller.

The interrupt service routine should poll all pending flags to determine the source of the interrupt. The service routine must then reset the pending bit of the interrupt being processed. This is normally done at the start of the interrupt service routine in order to avoid missing a fast second occurrence of the same interrupt. (Interrupt pending bits are NOT reset by hardware.) Maskable interrupts may be nested by setting the GIE bit at the start of or during the interrupt service routine. This procedure of nesting interrupts is not recommended except in very special cases. Great caution must be exercised in using nested interrupts because of the potential for stack overflow, as well as the possibility of over-writing registers used in the interrupt service routines.

Returning from a maskable interrupt service routine may be accomplished with any one of the following instructions; RET, RETSK or RETI. However, it is recommended that the RETI (return from interrupt) instruction be used. This instruction pops the last saved entry from the stack and places it in the Program Counter. In addition, it sets the GIE bit in order to enable further interrupts. The user may choose to set the GIE bit in software and use the RET (return from subroutine) or RETSK (return and skip) instruction.

3.3.1 Timer 1 Interrupt

Timer1 can be configured to generate an interrupt on one of two conditions. The PWM and External Event Counter modes of operation allow an interrupt on timer underflow. The Input Capture mode of operation allows an interrupt on a positive or negative edge transition on TIO (Pin G3). The same interrupt enable flag (ENTI) and interrupt pending flag (TPND) are used for both timer interrupts. These flags are located in the PSW register. Details on setting up the Timer1 interrupt are given in Chapter 4.

If the timer is not used, the TIO pin (G3) may function as an additional independent external interrupt. In order to set up this second external interrupt, the timer is placed in the Input Capture Mode. The timer control bit TC1 is used to select the edge polarity. The Timer Interrupt Enable (ENTI) and Timer Interrupt Pending (TPND) bits located in the PSW register are used as the enable and pending flags.

3.3.2 External Interrupt

The G0 pin on all COP800 devices may be used as an external interrupt input. If used as an external interrupt, the pin must be configured as an input as described in Chapter 2. The edge polarity of the interrupt may be selected by writing to the IEDG (External Interrupt Edged) bit located in the PSW register. Writing a zero selects a positive edge. Writing a one selects a negative edge. The interrupt is enabled when both the ENI

(External Interrupt Enable) bit and the GIE bit are set. The occurrence of an external interrupt will set the IPND (External Interrupt Pending) flag, also located in the PSW register.

3.4 SOFTWARE TRAP

The software trap interrupt is used to detect errors in program execution. These errors result from a variety of conditions including: brownouts, power transients, noise, runaway programs, over-popping of the stack and accessing program memory locations which are not physically present in the device.

A software trap occurs when a zero opcode instruction (INTR) is executed as part of the normal instruction sequence fetched from program memory. Generally, a zero opcode is only executed to force the acknowledgment of a hardware interrupt. In these cases, this opcode is not a part of the normal instruction sequence but is jammed into the instruction register by the interrupt logic. The execution of a zero opcode (INTR) when an interrupt is not pending is an error, and it is this error which actually results in a software trap. Such an error may occur when an instruction is fetched from beyond the available program memory space, when the stack is over-popped, or as a result of some transient condition. Reading from unavailable program memory always returns zeros (INTR). Thus, instruction fetches from these locations load zero opcodes into the instruction register, and thereby create a software trap. Over-popping a stack which has been initialized to the top of the available data memory space loads the Program Counter with FFFF Hex. This indirectly forces a software trap by loading the Program Counter with the address of an unavailable program memory location. If unused program memory is filled with all zeros, then a software trap will also occur if a runaway program inadvertently branches to an unused program memory location.

Software traps, like hardware interrupts, force a jump to program memory location 00FF Hex. However, software traps do not set an interrupt pending flag or reset the GIE bit. The code located at 00FF Hex can determine whether or not a software trap has occurred by polling all maskable interrupt pending flags. If no flags are set, it can be assumed that a software trap has occurred. Since the GIE bit is not reset upon the occurrence of a software trap, a software trap may be interrupted by another interrupt.

Whenever a software trap occurs, it is recommended that the user re-initialize the stack pointer and do a recovery procedure. The recovery procedure should be similar to a device RESET start-up but may not contain all of the same initialization procedures. The user should never simply execute a RET or RETI instruction to exit from a software trap. This is because the return address pushed onto the stack by the software trap is the address of the instruction that produced the error. An infinite loop of software traps is generated by returning to this instruction. The user may return to the instruction following the trap instruction by placing an RETSK at the end of the software trap service routine.

Chapter 4

TIMER

4.1 INTRODUCTION

The COP800 device contains a versatile 16-bit timer/counter that can satisfy a wide range of application requirements. The timer can be configured to operate in any of three modes:

- Pulse Width Modulation (PWM) mode: generates pulses of a specified width
- External event counter mode: counts occurrences of an external event
- Input capture mode: measures the elapsed time between occurrences of an external event

4.2 TIMER/COUNTER BLOCK

The timer/counter block (the section of the device containing the timer circuitry) consists of a 16-bit counter/timer register and an associated 16-bit autoload/capture register (designated RA). The timer and the associated autoload register are each organized as a pair of 8-bit memory-mapped registers. The timer bytes reside at addresses 00EA and 00EB, while the associated autoload register bytes reside at addresses 00EC and 00ED.

The timer/counter block uses one pin, designated TIO, to support the I/O requirements of the timer. The TIO feature is an alternate function of G3 (Port G, bit 3).

The timer can be started or stopped at any time under program control. When running, the timer counts down (decrements). Depending on the operating mode, the timer counts either instruction clock pulses (the clock used for executing instructions) or transitions on the TIO pin. Occurrences of the timer underflow (transition from 0000 to FFFF) can either generate an interrupt or toggle the TIO pin, also depending on the operating mode.

When timer interrupts are enabled, the source of the interrupt depends on the timer operating mode: either a timer underflow, or an input signal received on the TIO pin.

4.3 TIMER CONTROL BITS

The timer is controlled by six memory-mapped control bits in the PSW (Processor Status Word) and CNTRL (Control) registers of the CPU core. These bits control the operation of the timer by enabling or disabling the timer interrupt, by setting the operating mode, and by starting and stopping the timer. The control bits operate as described in Tables 4-1 and 4-2.

Table 4-1 Timer Control Bits

Register/Bit	Name	Function
PSW/Bit 5	TPND	Timer interrupt pending flag: 1 = Timer interrupt pending, 0 = Timer interrupt not pending
PSW/Bit 4	ENTI	Enable timer interrupt: 1 = Timer interrupt enabled, 0 = Timer interrupt disabled
CNTRL/Bit 7	TC3	Timer control bit 3 (see table 4-2)
CNTRL/Bit 6	TC2	Timer control bit 2 (see table 4-2)
CNTRL/Bit 5	TC1	Timer control bit 1 (see table 4-2)
CNTRL/Bit 4	TRUN	Timer run: 1 = Start timer, 0 = Stop timer

Table 4-2 Timer Mode Control Bits

CNTRL Bits 7-6-5	Operating Mode	T Interrupt	Timer Counts On
0-0-0	External event counter with autoload register	Timer underflow	TIO positive edge
0-0-1	External event counter with autoload register	Timer underflow	TIO negative edge
0-1-0	Not Allowed	—	—
0-1-1	Not Allowed	—	—
1-0-0	PWM: timer with autoload register	Timer underflow	Instruction clock
1-0-1	PWM: timer with autoload register; toggle TIO out	Timer underflow	Instruction clock
1-1-0	Timer with input capture register	TIO positive edge	Instruction clock
1-1-1	Timer with input capture register	TIO negative edge	Instruction clock

4.4 TIMER OPERATING MODES

The timer can be configured to operate in any one of three modes. Within each mode, there are options related to the use of the TIO pin.

The Pulse Width Modulation (PWM) mode can be used to generate precise pulses of known width on the TIO pin (configured as an output). The timer is clocked by the instruction clock. An underflow causes the timer register to be reloaded with the value in the RA register, and optionally, causes the TIO output to toggle.

The external event counter mode can be used to count occurrences of an external event. The timer is clocked by the signal appearing on the TIO pin (configured as an input). An underflow causes the timer register to be reloaded with the value in the RA register.

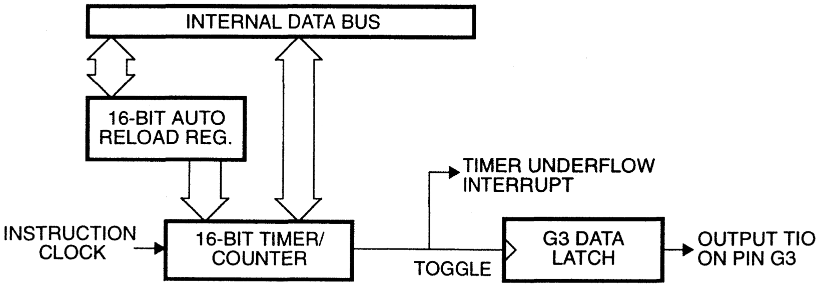
The input capture mode can be used to precisely measure the frequency of an external clock that is slower than the instruction clock, or to measure the elapsed time between external events. The timer is clocked by the instruction clock. A transition received on the TIO pin (configured as an input) causes a transfer of the timer contents to the RA register.

4.4.1 PWM Mode

In the Pulse Width Modulation (PWM) mode, the timer counts down at the instruction clock rate. When an underflow occurs, the contents of the RA register are transferred into the timer register, and counting proceeds downward from the loaded value. If the timer interrupt is enabled, an interrupt occurs with each underflow.

The timer can be configured to toggle the TIO output bit upon underflow. In this case, the width of pulses on the TIO pin are controlled by the value stored in the RA register.

A block diagram of the timer operating in the PWM mode is shown in Figure 4-1.



TSP-COP820-03

Figure 4-1 Timer in PWM Mode

The following steps can be used to operate the timer in the PWM mode. In this example, the TIO output pin is toggled with every timer underflow, and the "on" and "off" times for the TIO output are set to different values. (The TIO output can start out either high or low; follow the instructions shown in parentheses to start it low.)

1. Configure the TIO pin as an output by setting bit 3 in the Port G configuration register.
2. Initialize the TIO pin value to 1 (or 0) by setting (or clearing) the bit 3 in the Port G data register.
3. Load the PWM "on" (or "off") time into the timer register.
4. Load the PWM "off" (or "on") time into the RA register.
5. Set the timer control bits of the CNTRL register to select the PWM mode, and to toggle the TIO output with every timer underflow (see Table 4-2).
6. Set the TRUN (Timer Run) bit in the CNTRL register to start the timer.
7. After the timer underflows, update the RA register by writing the desired value for the next "on" or "off" time period. Either polling or interrupts can be used to synchronize loading of the RA register with the operation of the timer. To use interrupts, you must write the proper value to the PSW register before starting the timer: clear the TPND (Timer Interrupt Pending) flag, set the ENTI (Enable Timer Interrupt) bit, and set the GIE (Global Interrupt Enable) bit.

The selectable range for the PWM "on" and "off" times is 1 to 65,536 clock cycles. For a 10 MHz clock, this corresponds to a time range of 1 microsecond to 65.5 milliseconds. The pulse period ("on" plus "off" times) can then range from 2 microseconds to 131 milliseconds.

4.4.2 External Event Counter Mode

The external event counter mode is similar to the PWM mode, except that instead of counting instruction clock pulses, the timer counts transitions received on the TIO pin (configured as an input). The TIO pin should be connected to an external device that generates a pulse for each event to be counted.

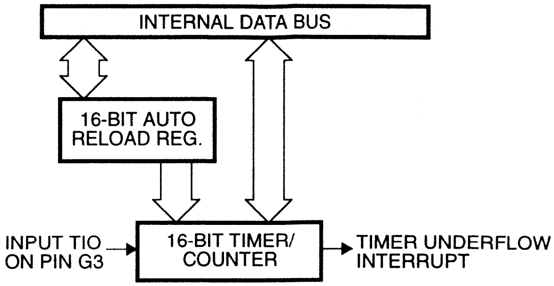
The timer can be configured to sense either positive-edge or negative-edge transitions on the TIO pin. The maximum frequency at which transitions can be sensed is one-half the frequency of the instruction clock.

As with the PWM mode, when an underflow occurs, the contents of the RA register are transferred into the timer register, and counting proceeds downward from the loaded value. If the timer interrupt is enabled, an interrupt occurs with each underflow.

A block diagram of the timer operating in the external event counter mode is shown in Figure 4-2.

The following steps can be used to operate the timer in the external event counter mode.

1. Configure the TIO pin as an input by clearing the bit 3 in the Port G configuration register.
2. Load the initial count into the timer register and also into the RA register. When this number of external events is detected, the counter will reach zero



TSP-COP820-04

Figure 4-2 Timer in External Event Counter Mode

although it will not overflow until the next event is detected. Therefore, to count N pulses, the value N-1 should be loaded into the timer and RA registers.

3. Set the timer control bits of the CNTRL register to select the external event counter mode, and to select the type of transition to be sensed on the TIO pin (positive-edge or negative-edge; see Table 4-2).
4. Set the TRUN (Timer Run) bit in the CNTRL register to start the timer.
5. The software should take whatever action is required when the timer underflows. Underflow can be detected either by polling the timer register or by using the timer interrupt. To use interrupts, you must write the proper value to the PSW register before starting the timer: clear the TPND (Timer Interrupt Pending) flag, set the ENTI (Enable Timer Interrupt) bit, and set the GIE (Global Interrupt Enable) bit.

4.4.3 Input Capture Mode

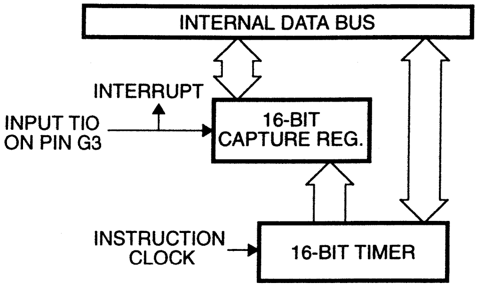
In the input capture mode, the timer counts down at the instruction clock rate. A transition received on the TIO pin (configured as an input) causes a transfer of the timer contents to the RA register. The values captured in the RA register at different times reflect the elapsed time between transitions on the TIO pin.

The timer can be configured to sense either positive-edge or negative-edge transitions. If the timer interrupt is enabled, the sensed transition on the TIO pin also triggers an interrupt. Timer underflows have no significance in this mode.

A block diagram of the timer operating in the input capture mode is shown in Figure 4-3.

The following steps can be used to operate the timer in input capture mode.

1. Configure the TIO pin as an input by clearing bit 3 in the Port G configuration register.
2. Set the timer control bits of the CNTRL to select the input capture mode, and to select the type of transition to be sensed on the TIO pin (positive-edge or negative-edge; see Table 4-2).



TSP-COP820-05

Figure 4-3 Timer in Input Capture Mode

3. Set the TRUN (Timer Run) bit in the CNTRL register to start the timer.

4. Each time a transition is sensed on the TIO pin, the contents of the timer register are transferred to the RA register, and an interrupt is triggered (if enabled). The interrupt service routine can record and compare the RA register contents to determine the elapsed time between events. To use interrupts, you must write the proper value to the PSW register before starting the timer: clear the TPND (Timer Interrupt Pending) flag, set the ENTI (Enable Timer Interrupt) bit, and set the GIE (Global Interrupt Enable) bit.

Chapter 5

MICROWIRE/PLUS

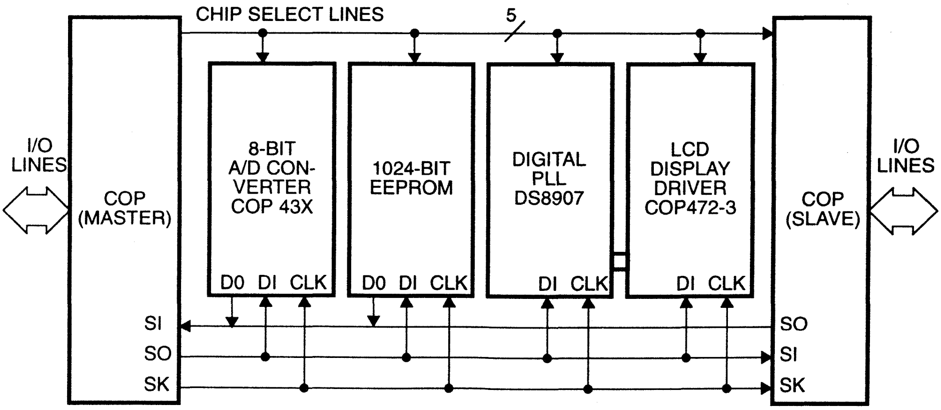
5.1 INTRODUCTION

MICROWIRE/PLUS™ is a synchronous serial communication system that allows the COP800 microcontroller to communicate with any other device that also supports the MICROWIRE/PLUS system. Examples of such devices include A/D converters, comparators, EEPROMs, display drivers, telecommunications devices, and other processors (e.g., HPC and COP400 processors). The MICROWIRE/PLUS serial interface uses a simple and economical 3-wire connection between devices.

Several MICROWIRE/PLUS devices can be connected to the same 3-wire system. One of these devices, operating in what is called the master mode, supplies the synchronous clock for the serial interface and initiates the data transfer. Another device, operating in what is called the slave mode, responds by sending (or receiving) the requested data. The slave device uses the master's clock for serially shifting data out (or in), while the master device shifts the data in (or out).

On the COP800 device, the three interface signals are called SI (Serial Input), SO (Serial Output), and SK (Serial Clock). To the master, SO and SK are outputs (connected to slave inputs), and SI is an input (connected to slave outputs).

The COP800 can operate either as a master or a slave, depending on how it is configured by the software. Figure 5-1 shows an example of how several devices can be connected together using the MICROWIRE/PLUS system, with the COP800 (on the left) operating as the master, and other devices operating as slaves. The protocol for selecting and enabling slave devices is determined by the system designer.

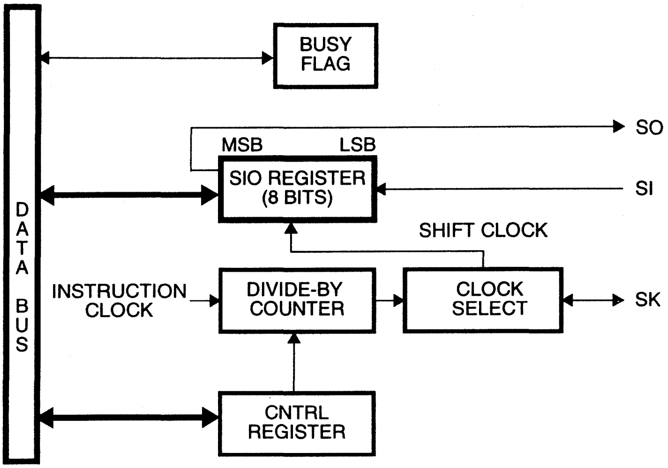


TSP-COP820-06

Figure 5-1 MICROWIRE/PLUS Example

5.2 THEORY OF OPERATION

Figure 5-2 is a block diagram illustrating the internal operation of the MICROWIRE/PLUS circuit of the COP800.



TSP-COP820-07

Figure 5-2 MICROWIRE/PLUS Circuit Block Diagram

An 8-bit shift register, called the SIO (Serial Input/Output) register, is used for both sending and receiving data. In either type of data transfer, bits are shifted left through the register. When a data byte is being sent, bits are shifted out through the SO output, most significant bit first. When a data byte is being received, bits are shifted in through the SI input, most significant bit first also.

The SIO register is memory-mapped in the microcontroller's data memory space, allowing the software to write a data byte to be sent, or to read a full data byte that has been received. The Busy flag in the PSW register indicates whether the SIO register is ready to be read or written. Instead of polling the Busy flag, you can use a carefully timed software loop to synchronize the reading or writing of the SIO register to completion of each 8-bit shift operation.

The software should write the SIO register only when the SK clock is low. A data byte is generally written at the end of an 8-bit shifting cycle, when the SK clock is low anyway, so this is generally not a problem. If the user inadvertently writes to the register when SK is high, unknown data may be placed in the register.

5.2.1 Timing

Timing of the MICROWIRE/PLUS interface is shown in Figure 5-3.

The SK clock signal is generated by the master device. Read and write operations are synchronized to this signal. When a data byte is being sent, the output data on SO is clocked out on the falling edge of the SK clock, as indicated by the solid arrows in the timing diagram. The first bit, however, becomes valid immediately after the SIO register

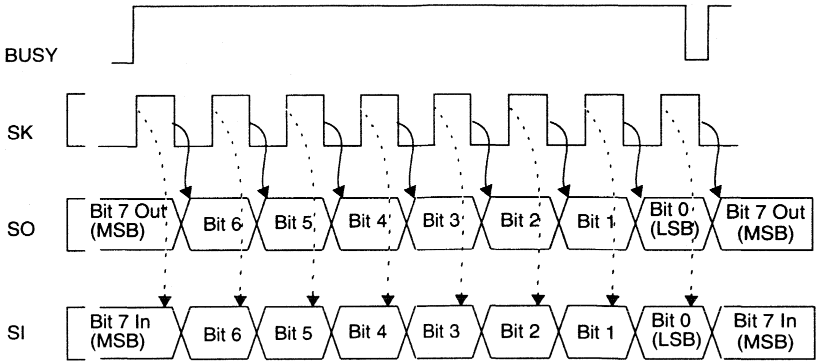


Figure 5-3 MICROWIRE/PLUS Interface Timing

is loaded with the data byte to be sent. When a data byte is being received, the input data on SI is sampled on the rising edge of the SK clock, as indicated by the dashed arrows in the timing diagram.

5.2.2 Port G Configuration

The three MICROWIRE/PLUS signals SO, SK, and SI are alternate functions of Port G pins G4, G5, and G6, respectively. To enable the use of these pins for the MICROWIRE/PLUS interface, the MSEL (MICROWIRE Select) bit of the CNTRL register must be set to 1. (The SL1 and SL0 bits, also in the CNTRL register, are used to control the SK clock speed in master mode, as described below.)

Port G must be properly configured for operation of the interface. This is accomplished by writing certain bit values to the Port G configuration register. Pin G4 (SO) should be configured as an output for sending data. Pin G5 (SK) should be configured as an output in master mode, or as an input in slave mode. G6 (SI) serves only as an input, so it need not be specifically configured as such. The Port G configuration register programming options are summarized in Table 5-1.

5.2.3 SK Clock Operation

When the COP800 operates in master mode, it generates the SK clock signal. A divide-by counter lowers the frequency of the instruction clock, producing an SK clock period that is 2, 4, or 8 times the period of the instruction clock. The divide-by factor is programmed by writing two bits to the CNTRL register, designated SL1 and SL0 (Select 1 and Select 0 bits), as indicated in Table 5-2.

The internal divide-by counter is reset when the MICROWIRE Busy flag (described below) goes to 1. Because of this, the divide-by counter always starts from 0 at the beginning of an 8-bit shift cycle, ensuring uniform SK clock pulses.

When the COP800 microcontroller operates in slave mode, the SK clock is generated by the external master device. In this case, SK is an input, and the SK clock-generating circuit of the COP800 is inactive.

Table 5-1 Port G Configuration Register Bits

Port G Config. Reg. Bits G5-G4	MICROWIRE Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
0-0	Slave, data in	TRI-STATE (unused)	SK Input	SI Input
0-1	Slave, data out and data in	SO Output	SK Input	SI Input
1-0	Master, data in	TRI-STATE (unused)	SK Output	SI Input
1-1	Master, data out and data in	SO Output	SK Output	SI Input

Table 5-2 Master Mode Clock Select Bits

SL1 (CNTRL Bit 1)	SL0 (CNTRL Bit 0)	SK Clock Period
0	0	2 times t_c
0	1	4 times t_c
1	X	8 times t_c

5.2.4 Busy Flag

A flag bit in the PSW (Processor Status Word) register indicates the status of the SIO shift register. To initiate an 8-bit shifting operation, set this bit to 1. Shifting then starts and continues automatically at the SK clock rate. With each shift, the high-order bit of the register is shifted out on SO (if enabled), and simultaneously, the low-order bit of the register is shifted in from SI.

When the 8-bit shifting operation is finished, the Busy flag is automatically reset to 0 by hardware. The software can determine whether or not shifting has been completed by polling this flag. When the flag is found to be 0, the software can write the next byte to be sent (or read the full byte just received) and then set the Busy flag to initiate transfer of the next byte.

The software can control the timing of the transfer by the setting and resetting the Busy flag. The handshaking protocol between the master and slave should ensure that the slave device is given enough time to respond after being enabled by the master. An example of a MICROWIRE/PLUS master/slave protocol is provided in the applications chapter.

The software program can reset the Busy bit directly by writing to the PSW register. This stops shifting immediately.

It is possible to eliminate the need for polling the Busy bit, thereby speeding up the transfer. This is accomplished by writing a software loop that executes in the exact amount of time necessary to allow an 8-bit shift operation. At the end of the loop, the software initiates the next 8-bit transfer, without checking the Busy bit. This is called the MICROWIRE "fast burst" mode. An example of this type of program is presented in the applications chapter.

Some external devices may require a continuous bit stream, without any pauses between bytes. This mode, called the MICROWIRE "continuous" mode, is also accomplished by writing a software loop that executes in a specific number of cycles. The clock divide-by factor must be 8. An example of this type of program is presented in the applications chapter.

When the COP800 operates in slave mode, the Busy flag should be set only when the SK clock signal (an input) is low. This is because the Busy bit is ANDed internally with the SK signal to produce the clock-shifting signal. If the Busy flag is set while SK is already high, the current SK pulse is gated in immediately, resulting in a clock pulse with an unknown width (perhaps very narrow), causing unreliable shifting.

5.3 MASTER MODE OPERATION EXAMPLE

When the COP800 operates in master mode, it generates the SK clock and initiates the transfer. The application software can perform a data transfer using the numbered steps shown below.

1. Write the proper value to the CNTRL register. To enable use of the Port G pins, set the MSEL bit. To set the divide-by factor for the SK clock, write the desired 2-bit value to the SL1 and SL0 bits (Table 5-2).
2. Write the proper value to the Port G configuration register, bits G5 and G4, to make the G5 (SK) pin an output and the G4 (SO) pin either TRI-STATE or an output, depending whether or not the COP800 is transmitting (Table 5-1).
3. If necessary, enable the desired slave device.
4. If sending data, write the data byte to the SIO register.
5. Set the Busy flag in the PSW register to initiate the transfer. Shifting proceeds automatically at the SK clock rate. The Busy flag is automatically reset upon completion of the 8-bit transfer.
6. Run in a loop and test the Busy flag for completion of the 8-bit transfer.
7. If receiving data, read the data byte in the SIO register.
8. Repeat steps 4 through 7 until all data bytes are transferred.

5.4 SLAVE MODE OPERATION EXAMPLE

When the COP800 operates in slave mode, the external master device generates the SK clock and initiates the transfer; SK is an input to the COP800. The application software can set up the COP800 device to allow a data transfer using the numbered steps shown below.

1. To enable use of the Port G pins, set the MSEL bit of the CNTRL register.
2. Write the proper value to the Port G configuration register to make the G5 (SK) pin an input and the G4 (SO) pin either TRI-STATE or an output, depending on whether or not the COP800 is transmitting (Table 5-1).
3. If sending data, write the data byte to the SIO register.
4. Set the Busy flag in the PSW register to allow the transfer. This should be done only when the SK signal is low. The handshaking protocol between the master and slave should ensure that the COP800 is given enough time to set the Busy flag before the data transfer starts. Once started, shifting proceeds at the SK clock rate. The Busy flag is automatically reset upon completion of the 8-bit transfer.
5. Run in a loop and test the Busy flag for completion of the 8-bit transfer.
6. If receiving data, read the data byte in the SIO register.
7. Repeat steps 3 through 6 until all data bytes are transferred.

Chapter 6

POWER SAVE MODE

6.1 INTRODUCTION

The COP800 supports a power-save mode of operation called the HALT mode. In this mode of operation, all internal processor activity stops and power consumption is reduced to a very low level. The processor can be forced to exit the HALT mode and resume normal operation at any time.

The fully static architecture of the COP800 allows the state of the microcontroller to be absolutely frozen. This is accomplished by stopping the internal clock of the device. In addition to stopping the internal clock, the controller stops the CKI pin from oscillating during the HALT mode if the R/C or Crystal clock is selected.

During normal operation, typical power consumption is in the range of 1 to 10 milliamperes. The actual power consumption for a device depends heavily on the clock speed and operating voltage used in an application. In the HALT mode, the device draws only a small amount of leakage current, plus any current necessary for driving the outputs. Typically, power consumption is reduced to less than 1 microampere. Since total power consumption is affected by the amount of current required to drive the outputs, all I/Os should be configured to draw minimal current, if possible, prior to entering the HALT mode. In order to reduce power consumption even further, the power supply (V_{CC}) can be reduced to a very low level during the HALT mode that guarantees the status of the RAM only. The allowed lower voltage level (Vr) is specified in the device datasheet.

There are two ways to enter the Halt mode. One method is to simply stop the processor clock (if the hardware implementation allows it). The other method is to set bit 7 of the Port G data register.

6.2 CLOCK-STOPPING METHOD

The clock-stopping method of entering the Halt mode can be used only if the hardware implementation of the processor clock allows it. If a crystal or R-C circuit is used, there is no practical way to stop the clock, and this method cannot be used. However, if the clock signal is generated externally and supplied to the CKI input, the external clock circuit can simply stop the clock at any time.

The clock signal at CKI can be stopped in either state (high or low). When the clock stops, the COP800 stops running but maintains all register and RAM contents. Power consumption is reduced to a very low level. However, when the clock starts running again, the processor begins running again from the point at which it had stopped.

6.3 PORT G METHOD

In the Port G method, the device enters the HALT mode under software control when the Port G data register bit 7 is set to 1. All processor action stops immediately, and power consumption is reduced to a very low level.

From this state, there are two ways to exit the Halt mode and resume normal operation. One method is to supply a low-to-high transition on the G7 input pin. The other method is to simply reset the device.

Using the G7 input pin is possible only if an external clock signal is supplied to CKI or an R-C circuit is being used. If a crystal circuit is being used, the G7/CKO pin is used as CKO, and is therefore unavailable for use as a HALT/Restart pin. If the G7 pin is available, a low-to-high transition on the pin takes the processor out of the Halt mode, and the program execution resumes from the point at which it stopped. In order to ensure accurate operation upon start-up of the device, the NOP (no-operation) instruction should follow the enter HALT instruction in the user's program.

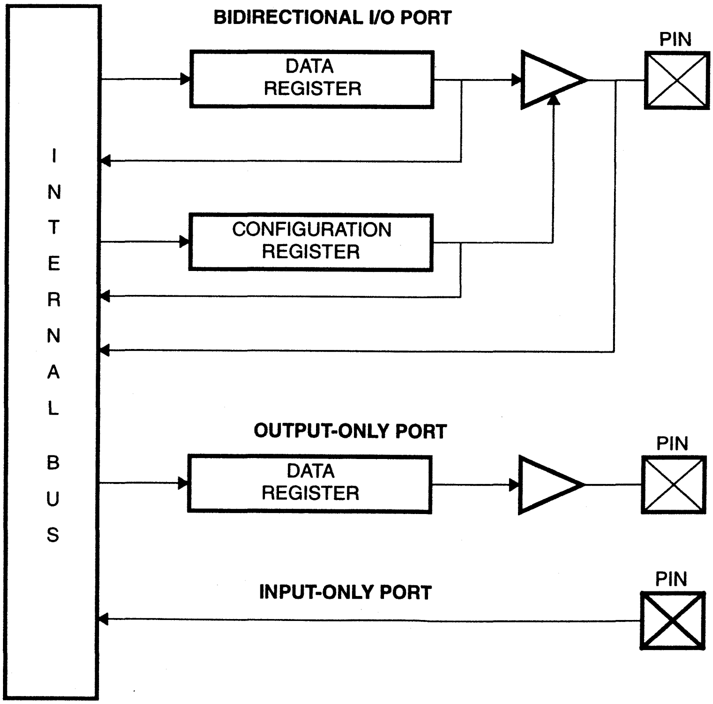
A device Reset, which is invoked by a low-level signal on the $\overline{\text{RESET}}$ input pin, takes the device out of the Halt mode and starts execution from address 0000H. The initialization software should determine what special action is needed, if any, upon start-up of the device. The initialization of all registers following a $\overline{\text{RESET}}$ exit from HALT is discussed in Section 2.6 and the device specific chapters.

Chapter 7

INPUT/OUTPUT

7.1 INTRODUCTION

All COP800 family devices have four dedicated input pins ($\overline{\text{RESET}}$, V_{CC}, GND, CKI) and at least one bidirectional I/O port. Additional bidirectional I/O ports, dedicated input ports, and dedicated output ports are available on higher pin-count packages for some devices. (Refer to the device specific chapters for additional information on available ports, packages, and pinouts.) The $\overline{\text{RESET}}$, V_{CC}, GND and CKI pins are used for reset, power supply, ground, and clock input, respectively. The bidirectional I/Os may be configured in software as Hi-Z input, weak pull-up, or push-pull output. These pins may be used as general purpose input/output pins or for selected alternate functions. The dedicated input and dedicated output ports may also be used as general purpose pins or for selected alternate functions. Individual port descriptions are given in the following sections of this chapter. Figure 7-1 contains a block diagram of the bidirectional, dedicated output, and dedicated input port types.



TSP-COP820-08

Figure 7-1 COP800 Port Structure

7.2 PORT C

Port C is not available on all COP800 devices. If present, Port C is a software configurable I/O port. All of the Port C pins are available for general purpose use. Three memory addresses are allocated to Port C. One address is used to read the port pins directly. The other two addresses are used to access the port configuration register and the port data register. The configuration and data registers' bits are used to set-up the individual pins of Port C as described in Section 2.3.3.

Any package which has a Port C with less than 8 pins contains unbonded pins. The user's software should write a "1" to the missing pins configuration register bits. This configures unbonded pins as outputs and reduces the leakage current of the part.

7.3 PORT D

Port D is not available on all COP800 devices. If present, Port D is a dedicated output port with one associated memory address. This address is used to access the port data register. A Port D pin may be individually configured to a logic high or low by writing a one or zero, respectively, to the associated data register bit. These pins are all available for general purpose use.

Port D outputs have high-sink drive capability. Refer to the COP800 datasheets for more information on the electrical specs of Port D.

Port D is preset high when $\overline{\text{RESET}}$ goes low, and the D2 pin is sampled. If D2 is held low during the reset state, the COP800 microcontroller enters a special mode of operation upon exiting the reset state. This special mode is used for testing purposes. In order to avoid entering this mode, the hardware designer should ensure D2 is not pulled low during reset.

7.4 PORT G

Port G is an input/output port which is available on all COP800 devices. The number of pins associated with this port varies according to the device and package. The G6 pin, if available, is always an input pin. The G7 pin is either an input or output depending on the oscillator mask option selected. With an RC oscillator or external clock, the pin is available as a general purpose input and HALT/Restart pin (see Chapter 6). With the crystal oscillator option, the pin is a dedicated clock output (CKO) pin. All other Port G pins are software configurable bi-directional input/outputs and are available for general purpose use.

Three memory addresses are assigned to this port. One address is used to read the actual port pins. The other two addresses are used to access the port data register and the port configuration register. The port registers' Bits 0-5 are used to individually configure the port pins G0 through G5 as described in Section 2.3.3. Bits 6 and 7 of the Port G configuration register, and Bit 6 of the Port G data register are reserved. These bits always read zero, and writing a value to these bits has no effect. Bit 7 of the Port G data register is used to place the COP800 microcontroller in HALT mode. (See Chapter 6.)

Most of the Port G pins are assigned optional alternate functions. The functions include but are not limited to: timer interface control, external interrupt, and MICROWIRE/PLUS interface. Alternate pin functions are listed in Section 7.7, and discussed in more detail in chapters devoted to specific COP800 features.

All Port G pins have Schmitt Triggers on their inputs. Any package which has a Port G with less than 8 pins contains unbonded pins. The user's software should write a "1" to the missing pins configuration register bits. This configures unbonded pins as outputs and reduces the leakage current of the part.

7.5 PORT I

Port I is not available on all COP800 devices. If present, Port I is a dedicated input port with one associated memory address. This read only address is used to access the input directly at the port pins.

All Port I pins are Hi-Z inputs, and must be pulled to a logic high or low externally. If a device has a Port I with less than 8 pins, the unavailable pins are unterminated. A read operation from these unterminated pins returns unpredictable values. The user should ensure that software takes this into account by either masking out these inputs or restricting the Port I accesses to bit operations only.

NOTE: Unterminated Port I pins draw power only when addressed (i.e., in short spikes).

7.6 PORT L

Port L is a bi-directional input/output port. The number of pins associated with this port varies according to the device and package. All Port L pins are available for general purpose use. Three memory addresses are allocated to Port L. One address is used to read the port pins directly. The other two addresses are used to access the port configuration register and the port data register. The configuration and data registers' bits are used to set-up the individual pins of Port L as described in Section 2.3.3.

In some COP800 devices, the Port L pins have been assigned optional alternate functions. Alternate pin functions, such as multi-input wakeup, are discussed in more detail in the device specific chapters.

Devices which support multi-input wakeup on the Port L pins have Schmitt Triggers on all Port L inputs. Some COP800 devices have high sink capabilities on Port L. Refer to the COP800 datasheets for more information on the Port L electrical characteristics.

7.7 ALTERNATE PORT FUNCTIONS

This section lists the alternate functions available on the Port G pins. For information on the alternate functions of pins in Ports C, D, I and L refer to the device specific chapters. Pins assigned alternate functions may be used in a general purpose manner or in their alternate function capacity.

PORT G

Port G has the following alternate pin functions on all COP800 devices:

G0 INTR (External Interrupt Input)

G1 No alternate function

G2 No alternate function

G3 Timer 1 I/O

G4 S0 (MICROWIRE/PLUS Serial Data Output)

G5 SK (MICROWIRE/PLUS Clock I/O)

G6 SI (MICROWIRE/PLUS Serial Data Input)

G7* Dedicated CKO (Clock Output) with Crystal Oscillator Mask Option or HALT/ Restart (Exit HALT Mode) with RC or External Oscillator Mask Option

Refer to the Interrupt, Timer, MICROWIRE/PLUS and Power Save Mode chapters for additional information on these pins.

*This pin's alternate function(s) cannot be enabled and disabled in software.

Chapter 8

INSTRUCTION SET

8.1 INTRODUCTION

This chapter defines the instruction set of the COP800 Basic Family members. It contains information about the instruction set features, addressing modes and types. In addition, it contains a detailed description of each COP800 instruction.

8.2 FEATURES

The strength of the COP800 instruction set is based on the following features:

- Majority of single-byte opcode instructions to minimize program size.
- One instruction cycle for the majority of single-byte instructions to minimize program execution time.
- Many single-byte, multiple function instructions such as DRSZ.
- Three memory mapped pointers; two for register indirect addressing, and one for the software stack.
- Sixteen memory mapped registers which allow an optimized implementation of certain instructions.
- Ability to set, reset and test any individual bit in data memory address space, including the memory mapped I/O ports and registers.
- Register-Indirect LOAD and EXCHANGE instructions with optional automatic post-incrementing or decrementing of the register pointer. This allows for greater efficiency (both in cycle time and program code) in loading, walking across and processing fields in data memory.
- Unique instructions to optimize program size and throughput efficiency. Some of these instructions are: DRSZ, IFBNE, DCOR, RETSK and RRC.

8.3 ADDRESSING MODES

The COP800 instruction set offers a variety of methods for specifying memory addresses. Each method is called an “addressing mode.” These modes are classified into two categories: “operand” addressing modes and “transfer-of-control” addressing modes. Operand addressing modes are the various methods of specifying an address for accessing (reading or writing) data. Transfer-of-control addressing modes are used in conjunction with “Jump” instructions to control the execution sequence of the software program.

8.3.1 Operand Addressing Modes

The operand of an instruction specifies what memory location is to be affected by that instruction. Several different operand addressing modes are available, allowing memory locations to be specified in a variety of ways. An instruction can specify an address directly by supplying the specific address, or indirectly by specifying a register pointer. The contents of the register (or in some cases, two registers) point to the desired memory location. In the "immediate" mode, the data byte to be used is contained in the instruction itself.

Each addressing mode has its own advantages and disadvantages with respect to flexibility, execution speed, and program compactness. Not all modes are available with all instructions. The Load (LD) instruction offers the largest number of addressing modes.

The available addressing modes are:

- Direct
- Register B or X Indirect
- Register B or X Indirect with Post-Incrementing/Decrementing
- Immediate
- Immediate Short
- Indirect from Program Memory

The addressing modes are described below. Each description includes an example of an assembly language instruction using the described addressing mode.

Direct. The memory address is specified directly as a byte in the instruction. In assembly language, the direct address is written as a numerical value (or a label that has been defined elsewhere in the program as a numerical value).

Example: Load Accumulator Memory Direct

LD A,05

Reg/Data Memory	Contents Before	Contents After
Accumulator	XX Hex	A6 Hex
Memory Location 0005 Hex	A6 Hex	A6 Hex

Register B or X Indirect. The memory address is specified by the contents of the B Register or X register (pointer register). In assembly language, the notation [B] or [X] specifies which register serves as the pointer.

Example: Exchange Memory with Accumulator, B Indirect

X A,[B]

Reg/Data Memory	Contents Before	Contents After
Accumulator	01 Hex	87 Hex
Memory Location 0005 Hex	87 Hex	01 Hex
B Pointer	05 Hex	05 Hex

Register B or X Indirect with Post-Incrementing/Decrementing. The relevant memory address is specified by the contents of the B Register or X register (pointer register). The pointer register is automatically incremented or decremented after execution, allowing easy manipulation of memory blocks with software loops. In assembly language, the notation [B+], [B-], [X+], or [X-] specifies which register serves as the pointer, and whether the pointer is to be incremented or decremented.

Example: Exchange Memory with Accumulator, B Indirect with Post-Increment

X A,[B+]

Reg/Data Memory	Contents Before	Contents After
Accumulator	03 Hex	62 Hex
Memory Location 0005 Hex	62 Hex	03 Hex
B Pointer	05 Hex	06 Hex

Immediate. The data for the operation follows the instruction opcode in program memory. In assembly language, the number sign character (#) indicates an immediate operand.

Example: Load Accumulator Immediate

LD A,#05

Reg/Data Memory	Contents Before	Contents After
Accumulator	XX Hex	05 Hex

Immediate Short. This is a special case of an immediate instruction. In the "Load B immediate" instruction, the 4-bit immediate value in the instruction is loaded into the lower nibble of the B register. The upper nibble of the B register is reset to 0000 binary.

Example: Load B Register Immediate Short

LD B,#7

Reg/Data Memory	Contents Before	Contents After
B Pointer	12 Hex	07 Hex

Indirect from Program Memory. This is a special case of an indirect instruction that allows access to data tables stored in Program Memory. In the "Load Accumulator Indirect" (LAID) instruction, the upper and lower bytes of the Program Counter (PCU and PCL) are used temporarily as a pointer to Program Memory. For purposes of accessing Program Memory, the contents of the Accumulator and PCL are exchanged. The data pointed to by the Program Counter is loaded into the Accumulator, and simultaneously, the original contents of PCL are restored so that the program can resume normal execution.

Example: Load Accumulator Indirect

LAID

Reg/Data Memory	Contents Before	Contents After
PCU	04 Hex	04 Hex
PCL	35 Hex	36 Hex
Accumulator	1F Hex	25 Hex
Memory Location 041F Hex	25 Hex	25 Hex

8.3.2 Transfer-of-Control Addressing Modes

Program instructions are usually executed in sequential order. However, "Jump" instructions can be used to change the normal execution sequence. Several transfer-of-control addressing modes are available to specify jump addresses.

A change in program flow requires a non-incremental change in the Program Counter contents. The Program Counter consists of two bytes, designated the upper byte (PCU) and lower byte (PCL). The most significant bit of PCU is not used, leaving 15 bits to address the program memory.

Different addressing modes are used to specify the new address for the Program Counter. The choice of addressing mode depends primarily on the distance of the jump. Farther jumps sometimes require more instruction bytes in order to completely specify the new Program Counter contents.

The available transfer-of-control addressing modes are:

- Jump Relative
- Jump Absolute
- Jump Absolute Long
- Jump Indirect

The transfer-of-control addressing modes are described below. Each description includes an example of a "Jump" instruction using a particular addressing mode, and the effect on the Program Counter of executing that instruction.

Jump Relative. In this 1-byte instruction, six bits of the instruction opcode specify the distance of the jump from the current program memory location. The distance of the jump can range from –31 to +32.

Example: Jump Relative

JP 0A

Reg	Contents Before	Contents After
PCU	02 Hex	02 Hex
PCL	05 Hex	0F Hex

Jump Absolute. In this 2-byte instruction, 12 bits of the instruction opcode specify the new contents of the Program Counter. The upper three bits of the Program Counter remain unchanged, restricting the new Program Counter address to the same 4-Kbyte address space as the current instruction. (This restriction is relevant only in devices using more than one 4-Kbyte program memory space.)

Example Jump Absolute

JMP 0125

Reg	Contents Before	Contents After
PCU	0C Hex	01 Hex
PCL	77 Hex	25 Hex

Jump Absolute Long. In this 3-byte instruction, 15 bits of the instruction opcode specify the new contents of the Program Counter.

Example: Jump Absolute Long

JMP 03625

Reg/Memory	Contents Before	Contents After
PCU	42 Hex	36 Hex
PCL	36 Hex	25 Hex

Jump Indirect. In this 1-byte instruction, the lower byte of the jump address is obtained from a table stored in program memory, with the Accumulator serving as the low order byte of a pointer into program memory. For purposes of accessing program memory, the contents of the Accumulator are written to PCL (temporarily). The data pointed to by the Program Counter (PCH/PCL) is loaded into PCL, while PCH remains unchanged.

Example: Jump Indirect

JID

Reg/Memory	Contents Before	Contents After
PCU	01 Hex	01 Hex
PCL	C4 Hex	32 Hex
Accumulator	26 Hex	26 Hex
Memory Location 0126 Hex	32 Hex	32 Hex

8.4 INSTRUCTION TYPES

The COP800 instruction set contains a fairly wide variety of instructions. The available instructions are listed below, organized into related groups.

Some instructions test a condition and skip the next instruction if the condition is not true. Skipped instructions are executed as no-operation (NOP) instructions.

Arithmetic Instructions

The arithmetic instructions perform binary arithmetic such as addition and subtraction, with or without the Carry bit.

Add (ADD)

Add with Carry (ADC)

Subtract (SUB)

Subtract with Carry (SUBC)

Increment (INC)

Decrement (DEC)

Decimal Correct (DCOR)

Clear Accumulator (CLR)

Set Carry (SC)

Reset Carry (RC)

Transfer-of Control Instructions

The transfer-of-control instructions change the usual sequential program flow by altering the contents of the Program Counter. The Jump to Subroutine instructions save the Program Counter contents on the stack before jumping; the Return instructions pop the top of the stack back into the Program Counter.

Jump Relative (JP)

Jump Absolute (JMP)

Jump Absolute Long (JMPL)

Jump Indirect (JID)

Jump to Subroutine (JSR)

Jump to Subroutine Long (JSRL)

Return from Subroutine (RET)

Return from Subroutine and Skip (RETSK)

Return from Interrupt (RETI)

Software Trap Interrupt (INTR)

Load and Exchange Instructions

The load and exchange instructions write byte values in registers or memory. The addressing mode determines the source of the data.

Load (LD)

Load Accumulator Indirect (LAID)

Exchange (X)

Logical Instructions

The logical instructions perform the basic logical operations AND, OR, and XOR (Exclusive OR). Other logical operations can be performed by combining these basic operations. For example, complementing is accomplished by exclusive-ORing the Accumulator with FF Hex.

Logical AND (AND)

Logical OR (OR)

Exclusive OR (XOR)

Accumulator Bit Manipulation Instructions

The Accumulator bit manipulation instructions allow the user to shift the Accumulator bits and to swap its two nibbles.

Rotate Right Through Carry (RRC)

Swap Nibbles of Accumulator (SWAP)

Memory Bit Manipulation Instructions

The memory bit manipulation instructions allow the user to set and reset individual bits in memory.

Set Bit (SBIT)

Reset Bit (RBIT)

Conditional Instructions

The conditional instructions test a condition. If the condition is true, the next instruction is executed in the normal manner; if the condition is false, the next instruction is skipped.

If Equal (IFEQ)

If Greater Than (IFGT)

If Carry (IFC)

If Not Carry (IFNC)

If Bit (IFBIT)

If B Pointer Not Equal (IFBNE)

Decrement Register and Skip if Zero (DRSZ)

No-Operation Instruction

The no-operation instruction does nothing, except to occupy space in the program memory and time in execution.

No-Operation (NOP)

8.5 INSTRUCTION DESCRIPTIONS

The COP800 Basic Family microcontrollers each contain 49 different instructions. Most of the arithmetic, comparison, and data transfer (load, exchange) instructions operate with three different addressing modes (register indirect with **B** pointer, memory direct, and immediate). These various addressing modes increase the instruction total to 75. The detailed instruction descriptions contain the following:

- Opcode mnemonic
- Instruction syntax with operand field descriptor
- Full instruction description
- Register level instruction description
- Number of instruction cycles
- Number of bytes in instruction
- Hexadecimal code for the instruction bytes

The following abbreviations represent the nomenclature used in the detailed instruction description and the COP800 cross-assembler:

A — Accumulator.

B — B Pointer, located in RAM register memory location 00FE.

[B] — Contents of RAM data memory location indicated by B pointer.

[B+] — Same as [B], except that B pointer is post-incremented.

[B-] — Same as [B], except that B pointer is post-decremented.

C — Carry flag, located in bit 6 of the PSW register at memory location 00EF Hex.

HC — Half Carry flag, located in bit 7 of the PSW register at memory location 00EF Hex.

MA — 8-bit memory address for RAM data store memory.

MD — Memory Direct, which may be represented by an implicit label (B, X, SP), a defined label (TEMP, COUNTER, etc.), or a direct memory address (12, 0EF, 027, etc., where a leading 0 indicates hexadecimal).

PC Program Counter (15 bits, with a program memory addressing range of 32768).

PCU Program Counter Upper, which contains the upper 7 bits of PC.

PCL Program Counter Lower, which contains the lower 8 bits of PC.

PSW Processor Status Word Register, found at memory location 00EF.

REG Selected Register (1 of 16) from the RAM data store memory at addresses 00F0-00FF.

REG# # of memory register to be used (# = 0-F hexadecimal).

symbol is used to indicate an immediate value, with a leading zero (0) indicating hexadecimal.

EXAMPLES:

#045 = immediate value of hexadecimal 45

#45 = immediate value of decimal 45

may also be used to indicate bit position, where # = 0-7

EXAMPLE:

RBIT #, [B]

SP Stack Pointer, located in RAM register memory location 00FD.

X X pointer, located in RAM register memory location 00FC.

[X] Contents of RAM data memory location indicated by the X pointer.

[X+] Same as [X], except that the X pointer is post-incremented.

[X-] Same as [X], except that the X pointer is post-decremented.

8.5.1 ADC— Add with Carry

Syntax:

a) ADC A,[B]

b) ADC A,#

c) ADC A,MD

Description:

The contents of

a) the data memory location referenced by the **B** pointer

b) the immediate value found in the second byte of the instruction

c) the data memory location referenced by the second byte of the instruction

are added to the contents of the accumulator, and the result is simultaneously incremented if the Carry flag is found previously set. The result is placed back in the accumulator, and the Carry flag is either set or reset, depending on the presence or absence of a carry from the result. Similarly, the Half Carry flag is either set or reset, depending on the presence or absence of a carry from the low-order nibble.

Operation:

A <- A + VALUE + C

C <- CARRY; HC <- HALF CARRY

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
ADC A,[B]	Register Indirect (B Pointer)	1	1	80
ADC A,#	Immediate	2	2	90/Imm #
ADC A,MD	Memory Direct	4	3	BD/MA/80

8.5.2 ADD — Add

Syntax:

a) ADD A,[B]

b) ADD A,MD

c) ADD A,#

Description:

The contents of the data memory location referenced by

a) the **B** pointer

b) the address in the second byte of the instruction

c) the immediate value found in the second byte of the instruction

are added to the contents of the accumulator, and the result is placed back in the accumulator. The Carry and Half Carry flags are not changed.

Operation: A <- A + VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
ADD A,[B]	Register Indirect (B Pointer)	1	1	84
ADD A,MD	Memory Direct	4	3	BD/MA/84
ADD A,#	Immediate	2	2	94/Imm.#

8.5.3 AND — And

Syntax:

a) AND A,[B]

b) AND A,#

c) AND A,MD

Description:

An AND operation is performed on corresponding bits of the accumulator and

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **AND** VALUE

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
AND A,[B]	Register Indirect (B Pointer)	1	1	85
AND A,#	Immediate	2	2	95/Imm.#
AND A,MD	Memory Direct	4	3	BD/MA/85

8.5.4 CLR — Clear Accumulator

Syntax: CLR A

Description: The accumulator is cleared to all zeros.

Operation: A <- 0

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
CLR A	Implicit	1	1	64

8.5.5 DCOR — Decimal Correct

Syntax: DCOR A

Description: This instruction when used following an ADC (add with carry) or SUBC (subtract with carry) instruction will decimal correct the result from the binary addition or subtraction. Note that the ADC instruction must be preceded with an ADD A, #066 (add hexadecimal 66) instruction for the decimal addition correction. This instruction assumes that the two operands are in BCD (Binary Coded Decimal) format and produces the result in the same BCD format. The Carry and Half Carry flags remain unchanged.

Operation: A (BCD FORMAT) <- A (BINARY FORMAT)

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
DCOR A	Implicit	1	1	66

8.5.6 DEC — Decrement Accumulator

Syntax: DECA

Description: This instruction decrements the contents of the accumulator and places the result back in the accumulator. The Carry and Half Carry flags remain unchanged.

Operation: A <- A - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DEC A	Implicit	1	1	8B

8.5.7 DRSZ REG# — Decrement Register and Skip if Result is Zero

Syntax: DRSZ REG#

Description: This instruction decrements the contents of the selected memory register (selected by #, where # = 0 to F) and places the result back in the same register. If the result is zero, the next instruction is skipped. This instruction is useful where it is desired to repeat an instruction sequence a given number of times. The desired number of times that the instruction sequence is to be executed is placed in a register, and a DRSZ instruction with that register is coded at the end of the sequence followed by a JP (Jump Relative) instruction that branches back to the start of the instruction sequence. The JP branch-back instruction is executed each time around the instruction sequence loop until the register count is decremented down to zero, at which time the JP instruction is skipped as the program branches (skips) out of the loop.

Operation:

```
REG <- REG - 1
IF (REG - 1) = 0,
THEN SKIP NEXT INSTRUCTION
```

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DRSZ REG#	Register Direct (Implicit)	3	1	C (REG#)

8.5.8 IFBIT — Test Memory Bit

Syntax:

a) IFBIT #,[B]

b) IFBIT #,MD

Description:

The selected bit (# = 0 to 7, with 7 being high-order) from the data memory location referenced by the

a) **B** pointer is tested.

b) address in the second byte of the instruction is tested.

If the selected bit is high (=1), then the next instruction is executed. Otherwise, the next instruction is skipped.

Operation:

IF BIT (#) SELECTED FROM MEMORY

IS EQUAL TO 0,

THEN SKIP NEXT INSTRUCTION

Instruction	Address Mode	Instruction Cycle	Bytes	Hex Op Code
IFBIT #,[B]	Register Indirect (B Pointer)	1	1	7#
IFBIT #,MD	Memory Direct	4	3	BD/MA/7#

8.5.9 IFBNE # — If B Pointer Not Equal

Syntax: IFBNE #

Description: If the low-order nibble of the **B** pointer is not equal to # (where # = 0 to F), then the next instruction is executed. Otherwise, the next instruction is skipped. This instruction is useful where the **B** pointer is walked across a data field as part of a closed loop instruction sequence. The IFBNE instruction is coded at the end of the sequence followed by a JP (Jump Relative) instruction that branches back to the start of the instruction sequence. The # coded with the IFBNE represents the next address beyond the data field. The **B** pointer instruction with post-increment or decrement of the pointer may be used in walking across the data field in either direction. The instruction sequence branches back and repeats until the low-order nibble of the **B** pointer is found equal to the # (representing the next address beyond the data field), at which time the JP instruction is skipped as the program branches (skips) out of the loop.

Operation: IF **B** POINTER LOW-ORDER NIBBLE EQUALS #,

THEN SKIP NEXT INSTRUCTION

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
IFBNE #	Implicit	1	1	4#

8.5.10 IFC — Test if Carry

Syntax: IFC

Description: The next Instruction is executed if the Carry flag is found set. Otherwise, the next instruction is skipped. The Carry flag is left unchanged.

Operation: IF NO CARRY (C = 0),

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFC	Implicit	1	1	88

8.5.11 IFEQ — Test if Equal

Syntax:

a) IFEQ A,[B]

b) IFEQ A,#

c) IFEQ A,MD

Description

a) The contents of the data memory location referenced by the **B** pointer are compared for equality with the contents of the accumulator.

b) The immediate value found in the second byte of the instruction is compared for equality with the contents of the accumulator.

c) The contents of the data memory location referenced by the address in the second byte of the instruction are compared for equality with the contents of the accumulator.

A successful equality comparison results in the execution of the next instruction. Otherwise, the next instruction is skipped.

Operation:

```
IF A ≠ VALUE
THEN SKIP NEXT INSTRUCTION
```

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFEQ A,[B]	Register Indirect (B Pointer)	1	1	82
IFEQ A,#	Immediate	2	2	92/Imm.#
IFEQ A,MD	Memory Direct	4	3	BD/MA/82

8.5.12 IFGT — Test if Greater Than

Syntax:

a) IFGT A,[B]

b) IFGT A,#

c) IFGT A,MD

Description:

The contents of the accumulator are tested for being greater than

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

A successful greater than test results in the execution of the next instruction. Otherwise, the next instruction is skipped.

Operation:

IF A ≤ VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
IFGT A,[B]	Register Indirect (B Pointer)	1	1	83
IFGT A,#	Immediate	2	2	93/Imm.#
IFGT A,MD	Memory Direct	4	3	BD/MA/83

8.5.13 IFNC — Test if No Carry

Syntax: IFNC

Description: The next instruction is executed if the Carry flag is found reset. Otherwise, the next instruction is skipped. The Carry flag is left unchanged.

Operation: IF CARRY (C=1),

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFNC	Implicit	1	1	89

8.5.14 INC — Increment Accumulator

Syntax: INC A

Description: This instruction increments the contents of the accumulator and places the result back in the accumulator. The Carry and Half Carry flags remain unchanged.

Operation: A <- A + 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
INC A	Implicit	1	1	8A

8.5.15 INTR — Interrupt (Software Trap)

Syntax: INTR

Description:

This zero opcode software trap instruction first stores its return address in the data memory software stack and then branches to program memory location 00FF. This memory location is the common switching point for all COP800 interrupts, both hardware and software. The program starting at memory location 00FF sorts out the priority of the various interrupts and then vectors to the correct interrupt service routine.

In order to save the return address, the contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. Then SP is again decremented to set up the software stack for the next interrupt or subroutine. The return address has now been saved on the software stack in data memory RAM.

The INTR instruction is not meant to be programmed explicitly, but rather to be automatically invoked when certain error conditions occur. The reading of undefined (non-existent) ROM program memory produces all zeros, which in turn invokes the INTR instruction. A similar software trap can be set up if the subroutine Stack Pointer (SP) is initialized to the data memory location at the top of user RAM space. Then if the software stack is ever overpopped (more subroutine or interrupt returns than calls), all ones will be returned from the undefined (non-existent) RAM. This will cause the program to return to the program address FFFF Hex, which in turn will read all zeros and once again invoke the software trap INTR instruction.

Two precautions must be observed when dealing with the software interrupt and its associated interrupt service routine. First, unlike the hardware interrupts, the software interrupt does not reset the GIE (Global Interrupt Enable) flag. Consequently, the COP800 microcontrollers can be interrupted by other interrupt sources while servicing the software interrupt. Second, a RETSK (return and skip) instruction should be used when returning from the software interrupt service routine, rather than the normal return from interrupt (RETI) instruction. The RETI instruction simply returns to the INTR software instruction itself, resulting in an infinite program loop.

Operation:

```
[SP] <- PCL
[SP - 1] <- PCU
[SP - 2] : SET UP FOR NEXT STACK REFERENCE
PC <- 0FF
```

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
INTR	Implicit	7	1	00

8.5.16 JID — Jump Indirect

Syntax: JID

Description: The JID instruction uses the contents of the accumulator to point to an indirect vector table of program addresses. The contents of the accumulator are transferred to PCL (Lower 8 bits of PC), after which the data accessed from the program memory location addressed by PC is transferred to PCL. The program then jumps to the program memory location accessed by PC. It should be observed that PCU (Upper 7 bits of PC) is never changed during the JID instruction, so that the Jump Indirect must jump to a location in the current program memory page of 256 addresses. However, if the JID instruction is located at the last address of the page, the PC counter will have already incremented over the page boundary, and both accesses to program memory (vector table and the new instruction) will be fetched from the next page of 256 bytes.

Operation: PCL <- A

PCL <- ROM (PCU,A)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JID	Indirect	3	1	A5

8.5.17 JMP — Jump Absolute

Syntax: JMP ADDR

Description: This instruction jumps to the programmed memory address. The value found in the lower nibble (4 bits) of the first byte of the instruction is transferred to the lower nibble of PCU (Upper 7 bits of PC), and then the value found in the second byte of the instruction is transferred to PCL (Lower 8 bits of PC). The program then jumps to the program memory location accessed by PC.

It should be noted that the upper 3 bits of PC (12-14) are not changed, so the Jump Absolute instruction must jump to an address located in the current 4-Kbyte program memory segment. However, if a JMP instruction is programmed in the last address of the memory segment, the PC counter will have already incremented over the memory segment boundary; therefore, the jump is to a memory location in the following 4-Kbyte memory segment.

Operation: PC11-8 <- HIADDR (LOW NIBBLE OF FIRST BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JMP ADDR	Absolute	3	2	2HIADDR/LOADDR

8.5.18 JMPL — Jump Absolute Long

Syntax: JMPL ADDR

Description: The JMPL instruction allows branching to anywhere in the 32-Kbyte program memory space. The values found in the second and third bytes of the instruction are transferred to PCU (Upper 7 bits of PC) and PCL (Lower 8 bits of PC) respectively. The program then jumps to the program memory location accessed by PC.

Operation: PC14-8 <- HIADDR (SECOND BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (THIRD BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JMPL ADDR	Absolute	4	3	AC/HIADDR/LOADDR

8.5.19 JP — Jump Relative

Syntax: JP DISP

Description: The relative displacement value found in the instruction opcode (all 8 bits) is added to the Program Counter (PC). The normal PC incrementation is also performed. The displacement value allows a branch back from 0 to 31 places (with the 0 representing an infinite closed loop branch to itself) and a branch forward from 2 to 32 places. A branch forward of 1 is not allowed, since this zero opcode conflicts with the INTR software trap instruction.

Operation: PC <- PC + DISP + 1 (DISP   0)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JP DISP	Relative	3	1	0, 1, E, F + DISP #

8.5.20 JSR — Jump Subroutine

Syntax: JSR ADDR

Description: This instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address. The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address has now been saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the value found in the lower nibble (4 bits of the first byte of the instruction) is transferred to the lower nibble of PCU, and the value found in the second byte of the instruction is transferred to PCL. The program then jumps to the memory location accessed by PC. It should be noted that the upper 3 bits of PC (12-14) are not changed, so the subroutine must be located in the current 4-Kbyte program memory segment. If a JSR instruction is programmed in the last address of the memory segment, however, the PC counter will have already incremented over the memory segment boundary; therefore, the subroutine must be located in the next memory segment.

Operation:

```
[SP] <- PCL
[SP - 1] <- PCU
[SP - 2]:  SET UP FOR NEXT STACK REFERENCE
PC11-8 <- HIADDR (LOW NIBBLE OF FIRST BYTE OF INSTRUCTION)
PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)
```

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSR ADDR	Absolute	5	2	3HIADDR/LOADDR

8.5.21 JSRL — Jump Subroutine Long

Syntax: JSRL ADDR

Description: The JSRL instruction allows the subroutine to be located anywhere in the 32-Kbyte program memory space. The instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address.

The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the values found in the second and third bytes of the instruction are transferred to PCU and PCL respectively. The program then jumps to the program memory location accessed by PC.

Operation:

```
[SP] <- PCL
[SP - 1] <- PCU
[SP - 2]: SET UP FOR NEXT STACK REFERENCE
PC14-8 <- HIADDR (SECOND BYTE OF INSTRUCTION)
PC7-0 <- LOADDR (THIRD BYTE OF INSTRUCTION)
```

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSRL ADDR	Absolute	5	3	AD/HIADDR/LOADDR

8.5.22 LAID — Load Accumulator Indirect

Address Mode: INDIRECT

Description: The LAID instruction uses the contents of the accumulator to point to a fixed data table stored in program memory. The data table usually represents a translation matrix, such as the input from a keyboard or the output to a display.

The contents of the accumulator are exchanged with the contents of PCL (Lower 8 bits of PC). The data accessed from the program memory location addressed by PC is then transferred to the accumulator. Simultaneously, the original contents of PCL are transferred back to PCL from the accumulator. It should be observed that PCU (Upper 7 bits of PC) is not changed during the LAID instruction, so that the load accumulator indirect along with the associated fixed data table must both be located in the current memory page of 256 bytes. However, if the LAID instruction is located at the last address of the page, the PC counter will have already incremented over the page boundary resulting in the operand being fetched from the next page. Consequently, in this instance, the fixed data table must reside in the next page of 256 bytes in the program memory.

Operation: A <- ROM (PCU, A)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LAID	Indirect	3	1	A4

8.5.23 LD — Load Accumulator

Syntax:

a) LD A,[B]

b) LD A,[B+]

c) LD A,[B-]

d) LD A,#

e) LD A,MD

f) LD A,[X]

g) LD A,[X+]

h) LD A,[X-]

Description:

a) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator.

b) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator, and then the **B** pointer is post-incremented.

c) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator, and then the **B** pointer is post-decremented.

d) The immediate value found in the second byte of the instruction is loaded into the accumulator.

e) The contents of the data memory location referenced by the address in the second byte of the instruction are loaded into the accumulator.

f) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator.

g) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator, and then the **X** pointer is post-incremented.

h) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator, and then the **X** pointer is post-decremented.

Operation: A <- VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD A,[B]	Register Indirect (B Pointer)	1	1	AE
LD A,[B+]	Register Indirect With Post-Incrementing B Pointer	2	1	AA
LD A,[B-]	Register Indirect With Post-Decrementing B Pointer	2	1	AB
LD A,#	Immediate	2	2	98/Imm.#
LD A,MD	Memory Direct	3	2	9D/MA
LD A,[X]	Register Indirect (X Pointer)	3	1	BE
LD A,[X+]	Register Indirect With Post-Incrementing X Pointe	3	1	BA
LD A,[X-]	Register Indirect With Post-Decrementing X Pointer	3	1	BB

8.5.24 LD — Load B Pointer

Syntax: LD B,# (# < 16)

Description: The one's complement of the value found in the lower nibble (4 bits) of the instruction is transferred to the lower-nibble position of the **B** pointer register, with the upper-nibble position being cleared to all zeros.

Operation: B3-0 <- (15 - #) (1's complement of #)

B7-4 <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD B,#	Short Immediate	1	1	5(15-#)

8.5.25 LD — Load Memory

Syntax:

a) LD [B],#

b) LD [B+],#

c) LD [B-],#

d) LD MD,#

Description:

a) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer.

b) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer, and then the **B** pointer is post-incremented.

c) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer, and then the **B** pointer is post-decremented.

d) The immediate value found in the third byte of the instruction is loaded into the data memory location referenced by the address in the second byte of the instruction.

Operation:

a) [B] <- #

b) [B] <- #; B <- B + 1

c) [B] <- #; B <- B - 1

d) MD <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD [B],#	Register Indirect/Immediate	2	2	9E/Imm.#
LD [B+],#	Register Indirect With Post-Incrementing/Immediate	2	2	9A/Imm.#
LD [B-],#	Register Indirect With Post-Decrementing/Immediate	2	2	9B/Imm.#
LD MD,#	Memory Direct/Immediate	3	3	BC/MA/Imm.#

8.5.26 LD — Load Register

Syntax: LD REG,#

Description: The immediate value found in the second byte of the instruction is loaded into the data memory register referenced by the low-order nibble of the first byte of the instruction.

Operation: REG <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD REG,#	Implicit/Immediate	3	2	D(REG#)/Imm.#

8.5.27 NOP — No Operation

Syntax: NOP

Description: No operation is performed by this instruction, so the net result is a delay of one instruction cycle time.

Operation: NO OPERATION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
NOP	Implicit	1	1	B8

8.5.28 OR — Or

Syntax:

a) OR A,[B]

b) OR A,#

c) OR A,MD

Description:

An OR operation is performed on corresponding bits of the accumulator with

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **OR** VALUE

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
OR A,[B]	Register Indirect (B Pointer)	1	1	87
OR A,#	Immediate	2	2	97/Imm.#
OR A,MD	Memory Direct	4	3	BD/MA/87

8.5.29 RBIT — Reset Memory Bit

Syntax:

a) RBIT #,[B]

b) RBIT #,MD

Description:

The selected bit (# = 0 to 7, with 7 being high-order) of the data memory location referenced by the

a) **B** pointer is reset to 0.

b) address in the second byte of the instruction is reset to 0.

Operation:

[Address:#] <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RBIT #,[B]	Register Indirect (B Pointer)	1	1	6(8 + #)
RBIT #,MD	Memory Direct	4	3	BD/MA/6(8+#)

8.5.30 RC — Reset Carry

Syntax: RC

Description: Both the Carry and Half Carry flags are reset to 0.

Operation: C <- 0

HC <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RC	Implicit	1	1	A0

8.5.31 RET — Return from Subroutine

Syntax: RET

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), after which SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program memory location accessed by PC.

Operation:

PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RET	Implicit	5	1	8E

8.5.32 RETI — Return from Interrupt

Syntax: RETI

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), and SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program memory location accessed by PC. The Global Interrupt Enable flag (GIE) is set to 1.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

GIE <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETI	Implicit	5	1	8F

8.5.33 RETSK — Return and Skip

Syntax: RETSK

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), and SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to and then skips the instruction in the program memory location accessed by PC.

Operation:

PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETSK	Implicit	5	1	8D

8.5.34 RRC — Rotate Accumulator Right Through Carry

Address Mode: RRC A

Description: The contents of the accumulator and Carry flag are rotated right one bit position, with the Carry flag serving as a ninth bit position linking the ends of the 8-bit accumulator. The previous carry is transferred to the high-order bit position of the accumulator. The low-order accumulator bit (A0) is transferred to both the Carry flag and the Half Carry flag.

Operation: C -> A7 -> A6 -> A5 -> A4 -> A3 -> A2 -> A1 -> A0 -> C

A0 -> HC

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RRC A	Implicit	1	1	B0

8.5.35 SBIT — Set Memory Bit

Syntax:

a) SBIT #,[B]

b) SBIT #,MD

Description:

The selected bit (# = 0 to 7, with 7 being high-order) of the data memory location referenced by the

a) **B** pointer is set to 1.

b) address in the second byte of the instruction is set to 1.

Operation:

[Address:#] <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SBIT #,[B]	Register Indirect (B Pointer)	1	1	7(8 + #)
SBIT #,MD	Memory Direct	4	3	BD/MA/7(8+#)

8.5.36 SC — Set Carry

Syntax: SC

Description: Both the Carry and Half Carry flags are set to 1.

Operation: C <- 1

HC <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SC	Implicit	1	1	A1

8.5.37 SUBC — Subtract with Carry

Syntax:

a) SUBC A,[B]

b) SUBC A,#

c) SUBC A,MD

Description:

a) The contents of the data memory location referenced by the **B** pointer are subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

b) The immediate value found in the second byte of the instruction is subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

c) The contents of the data memory location referenced by the address in the second byte of the instruction are subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

The result is placed back in the accumulator, and the Carry flag is either reset or set, depending on the presence or absence of a borrow from the result. Similarly, the Half Carry flag is either reset or set, depending on the presence or absence of a borrow from the low-order nibble.

This instruction is implemented by adding the one's complement of the subtrahend to the accumulator and then incrementing the result. Consequently, the borrow is the equivalent of the absence of carry and vice versa. Similarly, the half carry is the equivalent of the absence of half borrow and vice versa. A previous borrow (absence of previous carry) will inhibit the incrementation of the result.

Operation:

A <- A - VALUE - $\overline{C}$

C <- ABSENCE OF BYTE BORROW

HC <- ABSENCE OF LOW NIBBLE HALF BORROW

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SUBC A,[B]	Register Indirect (B Pointer)	1	1	81
SUBC A,#	Immediate	2	2	91/Imm.#
SUBC A,MD	Memory Direct	4	3	BD/MA/81

8.5.38 SWAP — Swap Nibbles of Accumulator

Syntax: SWAP A

Description: The upper and lower nibbles of the accumulator are exchanged.

Operation: A(7-4) <--> A(3 - 0)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SWAP A	Implicit	1	1	65

8.5.39 X — Exchange Memory with Accumulator

Syntax:

a) X A,[B]

b) X A,[B+]

c) X A,[B-]

d) X A,MD

e) X A,[X]

f) X A,[X+]

g) X A,[X-]

Description:

a) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator.

b) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator, and then the **B** pointer is post-incremented.

c) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator, and then the **B** pointer is post-decremented.

d) The contents of the data memory location referenced by the address in the second byte of the instruction are exchanged with the contents of the accumulator.

e) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator.

f) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator, and then the **X** pointer is post-incremented.

g) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator, and then the **X** pointer is post-decremented.

Operation:

a) A <-> [B]

b) A <-> B; B <- B + 1

c) A <-> B; B <- B - 1

d) A <-> MD

e) A <-> X;

f) A <-> X; X <- X + 1

g) A <-> X; X <- X - 1

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
X A,[B]	Register Indirect (B Pointer)	1	1	A6
X A,[B+]	Register Indirect With Post-Incrementing B Pointer	2	1	A2
X A,[B-]	Register Indirect With Post-Decrementing B Pointer	2	1	A3
X A,MD	Memory Direct	3	2	9C/MA
X A,[X]	Register Indirect (X Pointer)	3	1	B6
X A,[X+]	Register Indirect With Post-Incrementing X Pointer	3	1	B2
X A,[X-]	Register Indirect With Post-Decrementing X Pointer	3	1	B3

8.5.40 XOR — Exclusive Or

Syntax:

a) XOR A,[B]

b) XOR A,#

c) XOR A,MD

Description:

An XOR (Exclusive OR) operation is performed on corresponding bits of the accumulator with

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A XOR VALUE

Instruction	**Addressing Mode**	**Instruction Cycles**	**Bytes**	**Hex Op Code**
XOR A,[B]	Register Indirect (B Pointer)	1	1	86
XOR A,#	Immediate	2	2	96/Imm.#
XOR A,MD	Memory Direct	4	3	BD/MA/86

8.6 INSTRUCTION SET SUMMARY TABLES

8.6.1 Instruction Operations Summary

INSTR		FUNCTION	REGISTER OPERATION
ADD	A, MemI	Add	A <- A + MemI
ADC	A, MemI	Add with carry	A <- A + MemI + C, C <- Carry
SUBC	A, MemI	Subtract with carry	A <- A - MemI + C, C <- Carry
AND	A, MemI	Logical AND	A <- A and MemI
OR	A, MemI	Logical OR	A <- A or MemI
XOR	A, MemI	Logical Exclusive-OR	A <- A xor MemI
IFEQ	A, MemI	IF equal	Compare A and MemI, Do next if A = MemI
IFGT	A, MemI	IF greater than	Compare A and MemI, Do next if A > MemI
IFBNE	#	IF B not equal	Do next if lower 4 bits of B not = Imm
DRSZ	Reg	Decrement Reg, skip if zero	Reg <- Reg - 1, skip if Reg goes to zero
SBIT	#, Mem	Set bit	1 to Mem.bit (bit = 0 to 7 immediate)
RBIT	#, Mem	Reset bit	0 to Mem.bit (bit = 0 to 7 immediate)
IFBIT	#, Mem	If bit	If Mem.bit is true, do next instruction
X	A, Mem	Exchange A with memory	A <-> Mem
LD	A, MemI	Load A with memory	A <- MemI
LD	Mem, Imm	Load Direct memory Immed.	Mem <- Imm
LD	Reg, Imm	Load Register memory immed.	Reg <- Imm
X	A, [B±]	Exchange A with memory [B]	A <-> [B] (B <- B ± 1)
X	A, [X±]	Exchange A with memory [X]	A <-> [X] (X <- X ±1)
LD	A, [B±]	Load A with memory [B]	A <- [B] (B <- B ±1)
LD	A, [X±]	Load A with memory [X]	A <- [X] (X <- X ±1)
LD	[B±], Imm	Load memory immediate	[B] <- Imm (B <- B ±1)
CLRA		Clear A	A <- 0
INC	A	Increment A	A <- A + 1
DEC	A	Decrement A	A <- A - 1
LAID		Load A indirect from ROM	A <- ROM(PU, A)
DCOR	A	Decimal correct A	A <- BCD correction (follows ADC, SUBC)
RRC	A	Rotate right through carry	C -> A7 -> .... -> A0 -> C
SWAP	A	Swap nibbles of A	A7...A4 <-> A3...A0
SC		Set C	C <- 1
RC		Reset C	C <- 0
IFC		If C	If C is true, do next instruction
IFNC		If Not C	If C is not true, do next instruction
JMPL	Addr.	Jump absolute long	PC <- ii (ii = 15 bits, 0 to 32K)
JMP	Addr.	Jump absolute	PC11...PC0 <- i (i = 12 bits) PC15...PC12 remain unchanged
JP	Disp.	Jump relative short	PC <- PC + r (r is -31 to +32, not 1)
JSRL	Addr.	Jump subroutine long	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- ii
JSR	Addr.	Jump subroutine	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC11..PC0 <- ii
JID		Jump indirect	PL <- ROM(PU, A)
RET		Return from subroutine	SP + 2, PL <- [SP], PU <- [SP - 1]
RETSK		Return and skip	SP + 2, PL <- [SP], PU <- [SP - 1], Skip next instr.
RETI		Return from interrupt	SP + 2, PL <- [SP], PU <- [SP - 1], GIE <- 1
INTR		Generate an interrupt	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- 0FF
NOP		No operation	PC <- PC + 1

8.6.2 Bytes and Cycles Per Instruction

Table 8-1 Instructions Using A and C

INSTR	BYTES/ CYCLES
CLRA	1/1
INCA	1/1
DECA	1/1
LAID	1/3
DCOR	1/1
RRCA	1/1
SWAPA	1/1
SC	1/1
RC	1/1
IFC	1/1
IFNC	1/1

Table 8-2 Transfer of Control Instructions

INSTR	BYTES/ CYCLES
JMPL	3/4
JMP	2/3
JP	1/3
JSRL	3/5
JSR	2/5
JID	1/3
RET	1/5
RETSK	1/5
RETI	1/5
INTR	1/7
NOP	1/1

Table 8-3 Memory Transfer Instructions

INSTR	REGISTER INDIRECT		DIRECT	IMMEDIATE	REGISTER INDIRECT AUTO INCR & DECR	
	[B]	[X]			[B+, B-]	[X+, X-]
X A,[a]	1/1	1/3	2/3		1/2	1/3
LD A,[a]	1/1	1/3	2/3	2/2	1/2	1/3
LD B, Imm				1/1[b]		
LD B, Imm				2/3[c]		
LD Mem, Imm	2/2		3/3		2/2	
LD Reg, Imm			2/3			

a. Memory location addressed by B or X or directly
b. IF B < 16
c. IF B > 15

Table 8-4 Arithmetic and Logic Instruction

INSTR	[B]	DIRECT	IMMEDIATE
ADD	1/1	3/4	2/2
ADC	1/1	3/4	2/2
SUBC	1/1	3/4	2/2
AND	1/1	3/4	2/2
OR	1/1	3/4	2/2
XOR	1/1	3/4	2/2
IFEQ	1/1	3/4	2/2
IFGT	1/1	3/4	2/2
IFBNE	1/1		
DRSZ	1/1	1/3	
SBIT	1/1	3/4	
RBIT	1/1	3/4	
IFBIT	1/1	3/4	

Table 8-5 Opcodes

UPPER NIBBLE																LOWER NIBBLE
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
JP-15	JP-31	LD 0F0,#I	DRSZ 0F0	RRCA	RC	ADC A,#I	ADC A,[B]	IFBIT 0,[B]	*	LD B,#0F	IFBNE 0	JSR x000-x0FF	JMP x000-x0FF	JP+17	INTR	0
JP-14	JP-30	LD 0F1,#I	DRSZ 0F1	*	SC	SUBC A,#I	SUBC A,[B]	IFBIT 1,[B]	*	LD B,#0E	IFBNE 1	JSR x100-x1FF	JMP x100-x1FF	JP+18	JP+2	1
JP-13	JP-29	LD 0F2,#I	DRSZ 0F2	X A,[X+]	X A,[B+]	IFEQ A,#I	IFEQ A,[B]	IFBIT 2,[B]	*	LD B,#0D	IFBNE 2	JSR x200-x2FF	JMP x200-x2FF	JP+19	JP+3	2
JP-12	JP-28	LD 0F3,#I	DRSZ 0F3	X A,[X-]	X A,[B-]	IFGT A,#I	IFGT A,[B]	IFBIT 3,[B]	*	LD B,#0C	IFBNE 3	JSR x300-x3FF	JMP x300-x3FF	JP+20	JP+4	3
JP-11	JP-27	LD 0F4,#I	DRSZ 0F4	*	LAID	ADD A,#I	ADD A,[B]	IFBIT 4,[B]	CLRA	LD B,#0B	IFBNE 4	JSR x400-x4FF	JMP x400-x4FF	JP+21	JP+5	4
JP-10	JP-26	LD 0F5,#I	DRSZ 0F5	*	JID	AND A,#I	AND A,[B]	IFBIT 5,[B]	SWAPA	LD B,#0A	IFBNE 5	JSR x500-x5FF	JMP x500-x5FF	JP+22	JP+6	5
JP-9	JP-25	LD 0F6,#I	DRSZ 0F6	X A,[X]	X A,[B]	XOR A,#I	XOR A,[B]	IFBIT 6,[B]	DCORA	LD B,#09	IFBNE 6	JSR x600-x6FF	JMP x600-x6FF	JP+23	JP+7	6
JP-8	JP-24	LD 0F7,#I	DRSZ 0F7	*	*	OR A,#I	OR A,[B]	IFBIT 7,[B]	*	LD B,#08	IFBNE 7	JSR x700-x7FF	JMP x700-x7FF	JP+24	JP+8	7
JP-7	JP-23	LD 0F8,#I	DRSZ 0F8	NOP	*	LD A,#I	IFC	SBIT 0,[B]	RBIT 0,[B]	LD B,#07	IFBNE 8	JSR x800-x8FF	JMP x800-x8FF	JP+25	JP+9	8
JP-6	JP-22	LD 0F9,#I	DRSZ 0F9	*	*	*	IFNC	SBIT 1,[B]	RBIT 1,[B]	LD B,#06	IFBNE 9	JSR x900-x9FF	JMP x900-x9FF	JP+26	JP+10	9
JP-5	JP-21	LD 0FA,#I	DRSZ 0FA	LD A,[X+]	LD A,[B+]	LD [B+],#I	INCA	SBIT 2,[B]	RBIT 2,[B]	LD B,#05	IFBNE 0A	JSR xA00-xAFF	JMP xA00-xAFF	JP+27	JP+11	A
JP-4	JP-20	LD 0FB,#I	DRSZ 0FB	LD A,[X-]	LD A,[B-]	LD [B-],#I	DECA	SBIT 3,[B]	RBIT 3,[B]	LD B,#04	IFBNE 0B	JSR xB00-xBFF	JMP xB00-xBFF	JP+28	JP+12	B
JP-3	JP-19	LD 0FC,#I	DRSZ 0FC	LD Md,#I	JMPL	X A,Md	*	SBIT 4,[B]	RBIT 4,[B]	LD B,#03	IFBNE 0C	JSR xC00-xCFF	JMP xC00-xCFF	JP+29	JP+13	C
JP-2	JP-18	LD 0FD,#I	DRSZ 0FD	DIR	JSRL	LD A,Md	RETSK	SBIT 5,[B]	RBIT 5,[B]	LD B,#02	IFBNE 0D	JSR xD00-xDFF	JMP xD00-xDFF	JP+30	JP+14	D
JP-1	JP-17	LD 0FE,#I	DRSZ 0FE	LD A,[X]	LD A,[B]	LD [B],#I	RET	SBIT 6,[B]	RBIT 6,[B]	LD B,#01	IFBNE 0E	JSR xE00-xEFF	JMP xE00-xEFF	JP+31	JP+15	E
JP-0	JP-16	LD 0FF,#I	DRSZ 0FF	*	*	*	RETI	SBIT 7,[B]	RBIT 7,[B]	LD B,#00	IFBNE 0F	JSR xF00-xFFF	JMP xF00-xFFF	JP+32	JP+16	F

where: I is the immediate data — Md is a directly addressed memory location —* is an unused opcode

Chapter 9

COP820C/COP840C/COP880C

9.1 INTRODUCTION

The COP820C/840C/880C are members of the COPS microcontroller family. They are fully static parts, fabricated using double-metal silicon gate microCMOS technology. These low cost microcontrollers are each a complete microcomputer containing all system timing, interrupt logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include an 8-bit memory-mapped architecture, MICROWIRE/PLUS serial I/O, a 16-bit timer/counter with capture register and a multi-sourced interrupt. Each I/O pin has software selectable options to adapt the COP820C/840C/880C to specific applications. Several versions of the part are available that operate over different voltage and temperature ranges. Refer to the datasheet for more specific information. High throughput is achieved with an efficient instruction set operating at a rate of 1 microsecond per instruction.

This chapter discusses the device specifics of the COP820C/840C/880C microcontrollers. Information relevant to all COP800 Basic Family members is not covered in this chapter, but may be found in the first eight chapters of this manual. In this chapter, the term "COP880" refers to all COP880C packages, including the COP881C. "COP840" refers to all COP840C packages, including the COP842C. "COP820" refers to all COP820C packages, including the COP822C.

9.2 BLOCK DIAGRAM

The diagram in Figure 9-1 shows the basic functional blocks associated with the COP820/840/880. These blocks include the Arithmetic Logic Unit (ALU), Timer, MICROWIRE/PLUS, I/O ports, and on-chip memory.

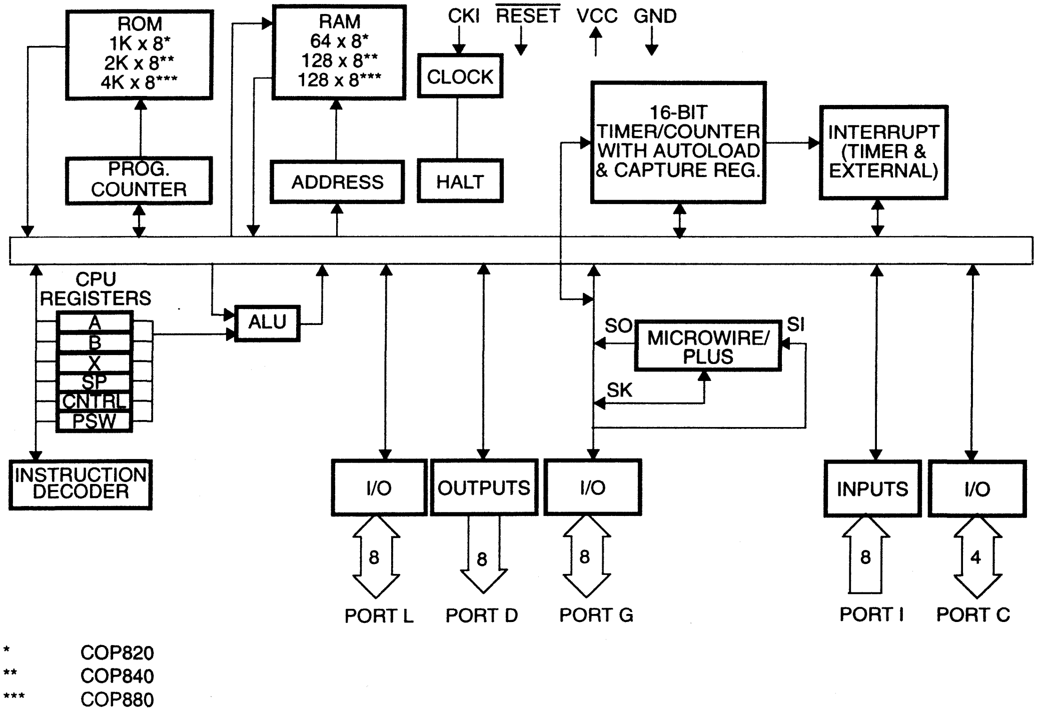
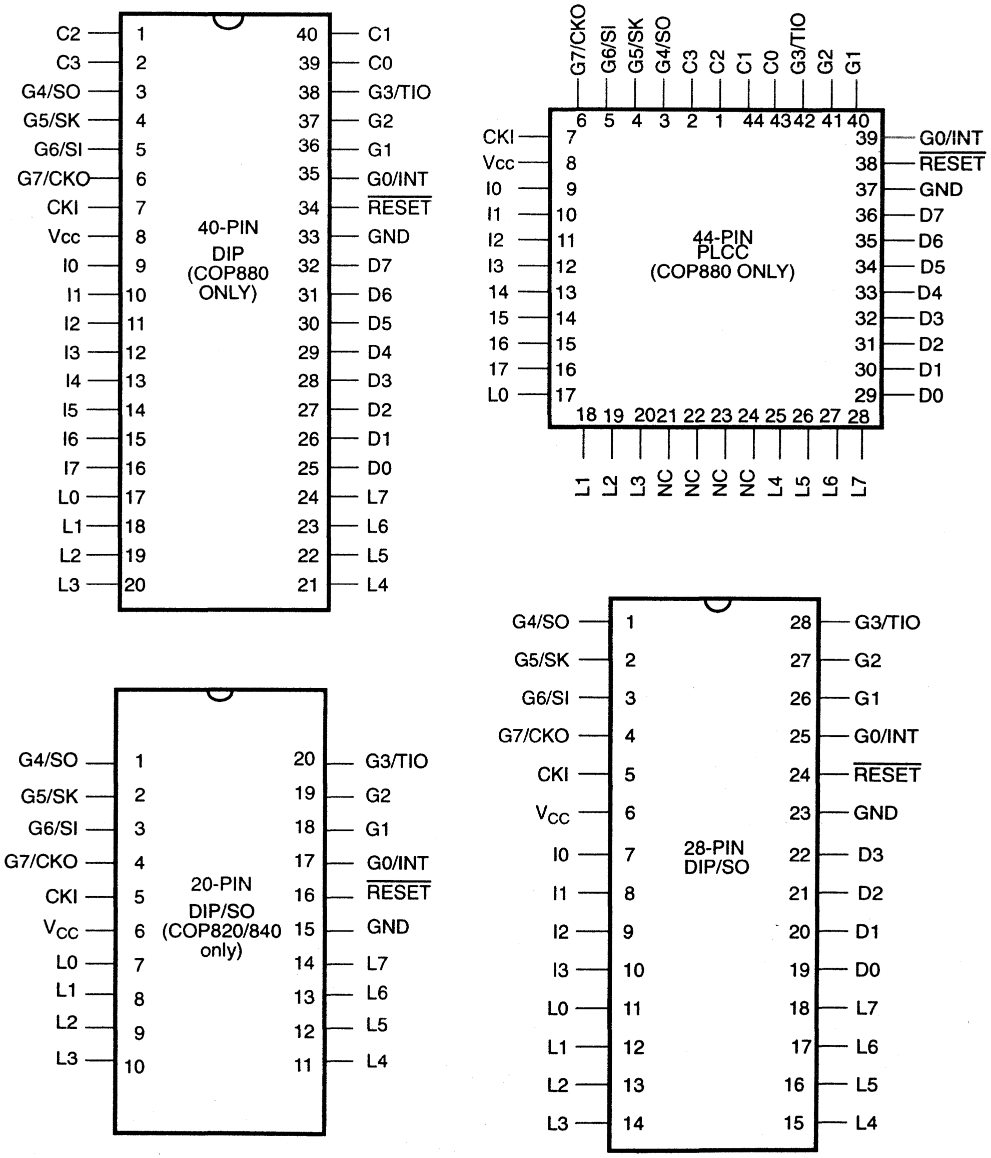


Figure 9-1 COP820/840/880 Block Diagram

9.3 DEVICE PINOUT/PACKAGES

The COP820 and COP840 are available in 20-pin DIP, 20-pin SO, 28-pin DIP, and 28-pin SO packages. The COP880 is available in 28-pin DIP, 28-pin SO, 40-pin DIP and 44-pin PLCC packages. Figure 9-2 shows the COP820/840/880 device package pinouts.

Refer to the COP820/840 and COP880 datasheets for more information on the device packages.



COP800-13

Figure 9-2 Device Package Pinouts

9.4 PIN DESCRIPTIONS

The COP820/840/880 have four dedicated function pins: V_{CC}, GND, CKI and $\overline{\text{RESET}}$. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. Table 9-1 lists the pin name, type, number and function of all COP820/840/880 signals.

9.5 INPUT/OUTPUT PORTS

The number of I/O ports available on the COP820/840/880 devices depends on package type. The COP820/840 20-pin packages have only a Port L and Port G. The 28-pin COP820/840/880 parts have a Port L, Port G, Port I and Port D. The 40- and 44-pin COP880 packages have a Port C in addition to the ports available on the 28-pin packages. All common COP800 ports are described in Chapter 7 of this manual. However, a brief description of each port is included in this section.

Port C, where available (40 pin DIP and 44 pin PLCC packages), is a 4-bit reconfigurable I/O port. The port is configured by writing to the Port C configuration and data registers as described in Section 2.5.3. Reading bits 4 - 7 of the Port C registers and input pins returns undefined data. It is the user's responsibility to mask out the upper four bits when reading Port C. This is accomplished by simply ANDing the Port C data with the value 000F Hex. This will ensure that the upper four bits of the Port C data are cleared. The Port C pins have not been assigned alternate functions.

Port D, where available, is a 4-bit (28 pin DIP/SO) or 8-bit (40 pin DIP and 44 pin PLCC) output only port. When writing an 8-bit quantity to devices which only have a 4-bit D Port, only the lower four bits are used. The Port D pins have no alternate functions.

Port G is an 8-bit reconfigurable I/O port. Pins 0 - 5 of the port are configured by writing to the Port G configuration and data registers as described in Section 2.5.3. Pin G6 is a dedicated input pin. Pin G7 is either an input or output, depending on the oscillator mask option selected. The Port G pins have the following alternate functions:

G0 INTR (External Interrupt Input)

G1 No alternate function

G2 No alternate function

G3 Timer 1 I/O

G4 S0 (MICROWIRE/PLUS Serial Data Output)

G5 SK (MICROWIRE/PLUS Clock I/O)

G6 SI (MICROWIRE/PLUS Serial Data Input)

G7 Dedicated CKO (Clock Output) with Crystal Oscillator Mask Option or HALT/Restart (Exit HALT Mode) with RC or External Oscillator Mask Option

Port I, where available, is a 4-bit (28 pin DIP/SO) or 8-bit (40 pin DIP and 44 pin PLCC) input-only port. All Port I pins are Hi-Z inputs. On the devices which only have a 4-bit

Table 9-1 COP820/840/880 Pin Assignments

PORT	TYPE	ALTERNATE FUNCTION	20 PIN DIP/SO	28 PIN DIP/SO	40 PIN DIP	44 PIN PLCC
L0	I/O		7	11	17	17
L1	I/O		8	12	18	18
L2	I/O		9	13	19	19
L3	I/O		10	14	20	20
L4	I/O		11	15	21	25
L5	I/O		12	16	22	26
L6	I/O		13	17	23	27
L7	I/O		14	18	24	28
G0	I/O	INTERRUPT	17	25	35	39
G1	I/O		18	26	36	40
G2	I/O		19	27	37	41
G3	I/O	TIO	20	28	38	42
G4	I/O	SO	1	1	3	3
G5	I/O	SK	2	2	4	4
G6	I	SI	3	3	5	5
G7	I/CKO	HALT RESTART	4	4	6	6
D0	O			19	25	29
D1	O			20	26	30
D2	O			21	27	31
D3	O			22	28	32
I0	I			7	9	9
I1	I			8	10	10
I2	I			9	11	11
I3	I			10	12	12
I4	I				13	13
I5	I				14	14
I6	I				15	15
I7	I				16	16
D4	O				29	33
D5	O				30	34
D6	O				31	35
D7	O				32	36
C0	I/O				39	43
C1	I/O				40	44
C2	I/O				1	1
C3	I/O				2	2
VCC			6	6	8	8
GND			15	23	33	37
CKI			5	5	7	7
$\overline{\text{RESET}}$			16	24	34	38

Port I, reading bits 4 - 7 of Port I will return undefined data. The user should mask out the upper four bits on these devices. No alternate functions have been assigned to the Port I pins.

Port L is an 8-bit reconfigurable I/O port. The port is configured by writing to the Port L configuration and data registers as described in Section 2.5.3. The Port L pins have no alternate functions.

9.6 PROGRAM MEMORY

The COP820C, COP840C and COP880C contain 1K bytes, 2K bytes and 4K bytes of program memory, respectively. The program memory may contain either instructions or data constants, and is addressed by the 15-bit program counter (PC). The program memory can be indirectly read by the LAID (Load Accumulator Indirect) instruction for table lookup of constant data. All program memory for the COP820/840/880 devices is mask-programmed ROM.

9.7 DATA MEMORY

The COP820 has 64 bytes of RAM data memory. These 64 bytes are memory mapped into two different locations. The first 48 bytes are resident from address 0000 to 002F Hex, while the remaining 16 bytes (containing the register memory) are located from address OOF0 to OOFF Hex.

The COP840 and COP880 have 128 bytes of RAM data memory. These 128 bytes are memory mapped into two different locations. The first 112 bytes are resident from address 0000 to 006F Hex, while the remaining 16 bytes are located from address OOF0 to OOFF Hex.

Refer to Chapter 2 for details on the data memory architecture.

9.8 REGISTER BIT MAPS

The COP820/840/880 devices have two registers that contain hardware control flags and bits. These registers, CNTRL and PSW, are located in the COP800 core and are described in the CORE REGISTERS section of this manual. The bit maps for these registers are shown below.

The PSW register bits are:

GIE	Global interrupt enable (enables interrupts)
ENI	External interrupt enable
BUSY	MICROWIRE/PLUS busy shifting flag
IPND	External interrupt pending
ENTI	Timer 1 interrupt enable

TPND	Timer 1 interrupt pending (timer underflow or capture edge)
C	Carry Flip/Flop
HC	Half-Carry Flip/Flop

Table 9-2 PSW Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	TPND	ENTI	IPND	BUSY	ENI	GIE

The timer and MICROWIRE/PLUS control register bits are:

SL1 & SLO	Selects the MICROWIRE/PLUS clock divide-by (00=2,01=4,1x=8)
IEDG	External interrupt edge polarity (0 = rising edge, 1 = falling edge)
MSEL	Selects G5 and G4 as MICROWIRE/PLUS signals SK and SO
TRUN	Used to start and stop the timer/counter (1 = run, 0 = stop)
TC1	Timer 1 Mode Control Bit
TC2	Timer 1 Mode Control Bit
TC3	Timer 1 Mode Control Bit

Table 9-3 CNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1	TC2	TC3	TRUN	MSEL	IEDG	SL1	SL0

9.9 MEMORY MAP

The COP820/840/880 is based on a memory-mapped architecture. All data memory, I/O ports, port registers and function registers are mapped into the data memory address space. Table 9-4 shows the organization of the data memory address space and the mapping of specific addresses. Read-only memory locations are noted in the table.

9.10 RESET

The following initializations are performed by the COP880/840/820 at reset:

PORT C:	TRI-STATE
PORT D:	LOGIC HIGH
PORT G:	TRI-STATE
PORT L:	TRI-STATE

Table 9-4 COP820/840/880 Memory Map

ADDRESS	CONTENTS
00 to 2F 30 to 6F 70 to 7F	On-chip RAM bytes On-chip RAM bytes (COP840/880 only) OR Unused RAM address space (reads as all 1's) (COP820 only) Unused RAM address space (reads as all 1's)
80 to BF	Reserved
C0 to CF	Reserved
D0 to DF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD to DF	On-chip I/O and registers Port L data register Port L configuration register Port L input pins (read only) Reserved Port G data register Port G configuration register Port G input pins (read only) Port I input pins (read only) Port C data register (COP880 only) Port C configuration register (COP880 only) Port C input pins (read only, (COP880 only) Reserved Port D Reserved
E0 to EF E0 to E8 E9 EA EB EC ED EE EF	On-chip functions and registers Reserved MICROWIRE/PLUS shift register (SIOR) Timer lower byte Timer upper byte Timer autoload register lower byte Timer autoload register upper byte CNTRL control register PSW register
F0 to FF FC FD FE	16 on-chip RAM bytes mapped as registers X register SP register B register

PC:	CLEARED
PSW and CNTRL:	CLEARED
B, X, SP:	UNKNOWN at power-on reset UNCHANGED at external reset
RAM:	UNKNOWN at power-on reset UNCHANGED at external reset
ACC and TIMER 1:	UNKNOWN at power-on reset UNKNOWN at external reset with Crystal oscillator clock option selected UNCHANGED at external reset with R/C or External oscillator clock options

9.11 MASK OPTION(S)

The COP820/840 and COP880 mask-selectable options are listed below. The options are programmed at the same time as the ROM pattern to provide the user with hardware flexibility.

9.11.1 COP820/840

Option 1: COP820C/COP840C CKI Input

=1 Normal Mode Crystal (CKI/10); CKO for crystal configuration

=2 Normal Mode External (CKI/10); CKO available as G7 input

=3 R/C (CKI/10); CKO available as G7 input

Option 2: COP820C/COP840C Bonding

=1 28-pin DIP

=2 N/A

=3 20-pin DIP

=4 20-pin SO

=5 28-pin SO

9.11.2 COP880

Option 1: COP880C/COP881C CKI Input

=1 Normal Mode Crystal (CKI/10); CKO for crystal configuration

=2 Normal Mode External (CKI/10); CKO available as G7 input

=3 R/C (CKI/10); CKO available as G7 input

Option 2: COP880C/COP881C Bonding

=1	44-pin PLCC
=2	40-pin DIP
=3	28-pin SO
=4	28-pin DIP

9.12 EMULATION DEVICES

The following chart shows the emulators available for the different COP820/840/880 packages. The emulators are discussed in detail in Appendix C.

Part Number	Emulator Package	Emulators (Type)
COP822C-XXX/N	20 DIP	COP822CMHD (MCM[a]) COP8782CN (OTP[b]) COP8782CJ (UV ERASABLE)
COP842C-XXX/N	20 DIP	COP842CMHD (MCM) COP8782CN (OTP) COP8782CJ (UV ERASABLE)
COP822C-XXX/WM COP842C-XXX/WM	20 SO	COP8782CWM (OTP) COP8782CMC (UV ERASABLE)
COP820C-XXX/N COP840C-XXX/N COP881C-XXX/N	28 DIP	COP881CMHD (MCM) COP8781N (OTP) COP8781CJ (UV ERASABLE)
COP820C-XXX/WM COP840C-XXX/WM COP881C-XXX/WM	28 LCC 28 SO 28 SO	COP881CMEA[c] (MCM) COP8781CWM (OTP) COP8781CMC (UV ERASABLE)
COP880C-XXX/N	40 DIP	COP880CMHD (MCM) COP8780CN (OTP) COP8780CJ (UV ERASABLE)
COP880C-XXX/V	44 LDCC 44 PLCC 44 LDCC	COP880CMHEL (MCM) COP8780CV (OTP) COP8780CEL (UV ERASABLE)

a. Multi-chip Module (UV Erasable)
b. One-Time Programmable
c. Same footprint as 28-pin SO

Chapter 10

COP8620C/COP8640C

10.1 INTRODUCTION

The COP8640C/8620C are members of the COPS microcontroller family. They are fully static parts, fabricated using double-metal silicon gate microCMOS technology. These low-cost microcontrollers are each a complete microcomputer containing all system timing, interrupt logic, ROM, RAM, EEPROM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include an 8-bit memory-mapped architecture, MICROWIRE/PLUS serial I/O, a 16-bit timer/counter with capture register and a multi-sourced interrupt. Each I/O pin has software selectable options to adapt the COP8640C/8620C to specific applications. The part operates over a voltage range of 4.5V to 6.0V. Certain family members (COP86L20/COP86L40) operate over a voltage range of 2.5V to 6.0V. Refer to the datasheet for more specific information. High throughput is achieved with an efficient, regular instruction set operating at a rate of 1 microsecond per instruction.

This chapter discusses the device specifics of the COP8640C/8620C microcontrollers. Information relevant to all COP800 Basic Family members is not covered in this chapter, but may be found in the first eight chapters of this manual. In this chapter, the term "COP8640" refers to all COP8640C packages, including the COP8642C. "COP8620" refers to all COP8620C packages, including the COP8622C.

10.2 BLOCK DIAGRAM

The diagram in Figure 10-1 shows the basic functional blocks associated with the COP8620/8640. These blocks include the Arithmetic Logic Unit (ALU), Timer, MICROWIRE/PLUS, I/O ports, and on-chip memory.

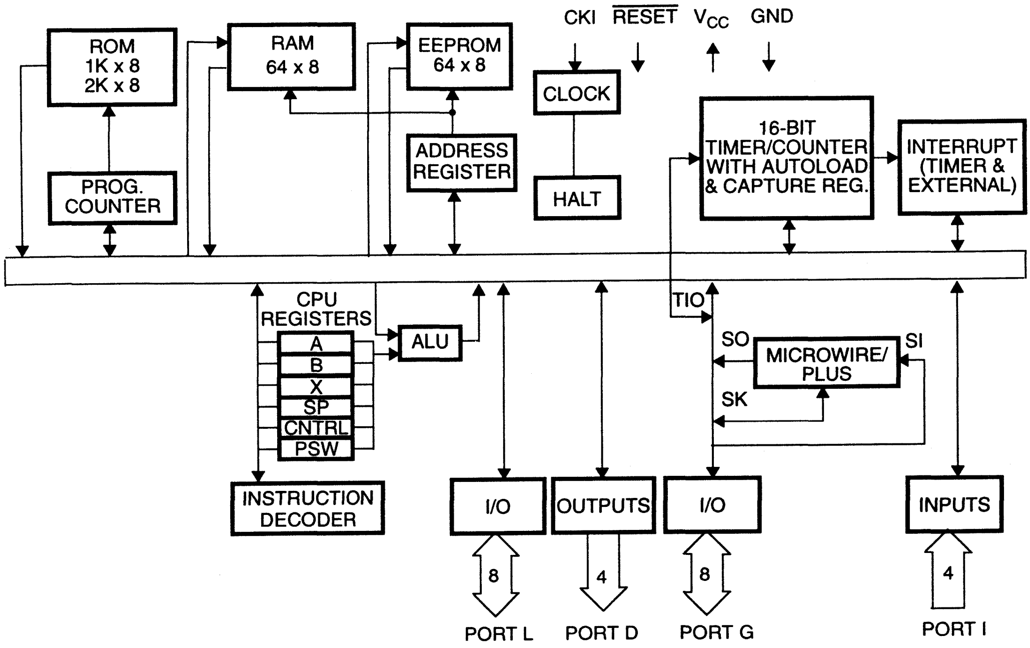
10.3 DEVICE PINOUT/PACKAGES

The COP8620 and COP8640 are available in 20-pin DIP, 20-pin SO, 28-pin DIP, and 28-pin SO packages. Figure 10-2 shows the COP8620/8640 device package pinouts.

Refer to the COP8620/8640 datasheets for more information on the device packages.

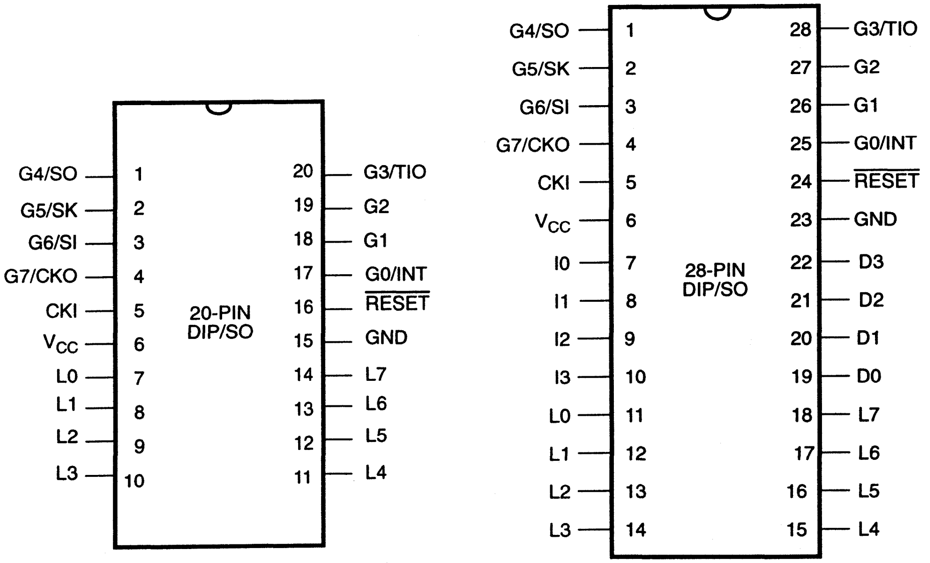
10.4 PIN DESCRIPTIONS

The COP8620/8640 have four dedicated function pins: V_{CC}, GND, CKI and $\overline{\text{RESET}}$. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as



COP800-14

Figure 10-1 COP8620/8640 Block Diagram



COP800-15

Figure 10-2 Device Package Pinouts

the master reset input, and CKI is used as a dedicated clock input. Table 10-1 lists the pin name, type, number and function of all COP8620/8640 signals.

Table 10-1 COP8620/8640 Pin Assignments

PORT	TYPE	ALTERNATE FUNCTION	20 PIN DIP/SO	28 PIN DIP/SO
L0	I/O		7	11
L1	I/O		8	12
L2	I/O		9	13
L3	I/O		10	14
L4	I/O		11	15
L5	I/O		12	16
L6	I/O		13	17
L7	I/O		14	18
G0	I/O	INTERRUPT	17	25
G1	I/O		18	26
G2	I/O		19	27
G3	I/O	TIO	20	28
G4	I/O	SO	1	1
G5	I/O	SK	2	2
G6	I	SI	3	3
G7	I/CKO	HALT RESTART	4	4
D0	O			19
D1	O			20
D2	O			21
D3	O			22
I0	I			7
I1	I			8
I2	I			9
I3	I			10
V_{CC}			6	6
GND			15	23
CKI			5	5
RESET			16	24

10.5 INPUT/OUTPUT PORTS

The number of I/O ports available on the COP8620/8640 devices depends on package type. The COP8620/8640 20-pin packages have only a Port L and Port G. The 28-pin COP8620/8640 parts have a Port L, Port G, Port I and Port D. All common COP800 ports are described in Chapter 7 of this manual. However, a brief description of each port is included in this section.

Port D, where available (28 pin DIP/SO packages), is a 4-bit output only port. When writing an 8-bit quantity to this port, only the lower four bits are used. The Port D pins have no alternate functions.

Port G is an 8-bit reconfigurable I/O port. Pins 0 - 5 of the port are configured by writing to the Port G configuration and data registers as described in Section 2.5.3. Pin G6 is a dedicated input pin. Pin G7 is either an input or output depending on the oscillator mask option selected. The Port G pins have the following alternate functions:

G0	INTR (External Interrupt Input)
G1	No alternate function
G2	No alternate function
G3	Timer 1 I/O
G4	S0 (MICROWIRE/PLUS Serial Data Output)
G5	SK (MICROWIRE/PLUS Clock I/O)
G6	SI (MICROWIRE/PLUS Serial Data Input)
G7	Dedicated CKO (Clock Output) with Crystal Oscillator Mask Option or HALT/ Restart (Exit HALT Mode) with RC or External Oscillator Mask Option

Port I, where available (28 pin DIP/SO packages), is a 4-bit input-only port. All Port I pins are Hi-Z inputs. Reading bits 4 - 7 of Port I will return undefined data. The user should mask out the upper four bits on these devices. No alternate functions have been assigned to the Port I pins.

Port L is an 8-bit reconfigurable I/O port. The port is configured by writing to the Port L configuration and data registers as described in Section 2.5.3. The Port L pins have no alternate functions.

10.6 PROGRAM MEMORY

The COP8620C and COP8640C contain 1K bytes and 2K bytes of program memory, respectively. This portion of memory contains the program instructions. The program memory can also contain look-up tables. The program memory is addressed by the 15-bit program counter (PC). It can be indirectly read by the LAID instruction for table lookup. All program memory for the COP8620/8640 devices is mask-programmed ROM.

10.7 DATA MEMORY

The data memory address space on the COP8620/8640 includes on-chip RAM, EEPROM, I/O, and registers. Data memory is addressed directly by an instruction or indirectly through B, X, and SP pointers.

The COP8620/8640 has 64 bytes of RAM. The first 48 bytes are mapped from locations 0000 Hex through 002F Hex. The remaining 16 bytes are mapped from locations 00F0 Hex through 00FF Hex. See Chapter 2 for details on the data memory architecture.

In addition to the 64 bytes of RAM, the COP8620/8640 provides 64 bytes of on-chip EEPROM for nonvolatile data storage. This EEPROM memory is mapped from location 0080 Hex through 00BF Hex. This is very useful for applications that require data to be maintained when there is no power. The data EEPROM can be read from and written to in exactly the same way as the RAM. All instructions that perform read and write operations on the RAM work similarly upon the data EEPROM. The data EEPROM contains 0000 Hex (all 0's) when shipped from the factory.

A data EEPROM write cycle is initiated by an instruction which accesses a location within the EEPROM such as X, LD, SBIT, and RBIT. The EE memory support circuitry sets the BsyERAM flag in the EECR register immediately upon beginning a data EEPROM write cycle. It will automatically be reset by the hardware at the end of the data EEPROM write cycle. The application program should test the BsyERAM flag before attempting a write operation to the data EEPROM. The following example illustrates how this can be accomplished:

```
TEST:  IFBIT  #BsyERAM, EECR

       JP     TEST

       LD     090, #0FF
```

In this example, the program continuously checks the state of the BsyERAM bit. Once the bit is reset, the program continues and writes to the EEPROM memory. (In this case, the value 00FF Hex is written into memory location 0090 Hex.)

A second EEPROM write operation while a write operation is still in progress will be ignored and the Werr flag in the EECR register will be set to indicate the error status.

10.8 EECR AND EE SUPPORT CIRCUITRY

The EEPROM portion of the COP8620/8640 contains EE support circuitry. This circuitry is needed to generate the high voltage programming pulses required to write the EEPROM memory. The support circuitry enables the user's program to read and write from EEPROM memory as if it were ordinary RAM. An EEPROM cell in the erase state is read out as a zero and the written state as a one. The EECR register provides control and status functions for the EE portion of the data memory. The EECR register has the following format:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	X	X	0	BsyERAM	Flag	0	Werr

Bit 7	=	X	Don't care. Cannot be used as general purpose flag bit.
Bit 6	=	X	Don't care. Cannot be used as general purpose flag bit.
Bit 5	=	X	Don't care. Cannot be used as general purpose flag bit.
Bit 4	=	0	Read-only bit. Always reads as "0".

Bit 3 = 1 Read-only bit. Set to "1" when EEPROM is being written to. It is automatically reset upon completion of the write operation. This bit is not cleared by a reset. If a reset occurs during a write operation, the BsyERAM bit will not be cleared until the write cycle is completed.

= 0 Read-only bit. Reads as a "0" when nothing is being written to the EEPROM.

Bit 2 = X Read/Write bit. This bit can be used as a general purpose flag.

Bit 1 = 0 Read-only bit. Always reads as "0".

Bit 0 = 1 Reading a "1" from this bit indicates that an attempt was made to write to the EEPROM while a previous write cycle was still in progress. Werr is a read/write bit and is cleared by writing a "0" to it.

= 0 Reading a "0" from this bit indicates that no error was encountered during a write cycle.

10.9 REGISTER BIT MAPS

The COP8620/8640 devices have two registers that contain hardware control flags and bits. These registers, CNTRL and PSW, are located in the COP800 core and are described in the CORE REGISTERS section of this manual. The bit maps for these registers are shown below.

The PSW register bits are:

GIE	Global interrupt enable (enables interrupts)
ENI	External interrupt enable
BUSY	MICROWIRE/PLUS busy shifting flag
IPND	External interrupt pending
ENTI	Timer 1 interrupt enable
TPND	Timer 1 interrupt pending (timer underflow or capture edge)
C	Carry Flip/Flop
HC	Half-Carry Flip/Flop

Table 10-2 PSW Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	TPND	ENTI	IPND	BUSY	ENI	GIE

The timer and MICROWIRE/PLUS control register bits are:

SL1 & SLO Select the MICROWIRE/PLUS clock divide-by (00=2,01=4,1x=8)

IEDG External interrupt edge polarity (0 = rising edge, 1 = falling edge)

MSEL Selects G5 and G4 as MICROWIRE/PLUS signals SK and SO

TRUN Used to start and stop the timer/counter (1 = run, 0 = stop)

TC1 Timer 1 Mode Control Bit

TC2 Timer 1 Mode Control Bit

TC3 Timer 1 Mode Control Bit

Table 10-3 CNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1	TC2	TC3	TRUN	MSEL	IEDG	SL1	SL0

10.10 MEMORY MAP

The COP8620/8640 is based on a memory mapped architecture. All data memory, I/O ports, port registers and function registers are mapped into the data memory address space. Table 10-4 shows the organization of the data memory address space and the mapping of specific addresses. Read-only memory locations are noted in the table.

Table 10-4 COP8620/8640 Memory Map

ADDRESS	CONTENTS
00 to 2F	On-chip RAM bytes
30 to 7F	Unused RAM address space (Reads as all 1's)
80 to BF	On-chip EEPROM (64 bytes)
C0 to CF	Reserved
D0 to DF	On-chip I/O and registers
D0	Port L data register
D1	Port L configuration register
D2	Port L input pins (read only)
D3	Reserved
D4	Port G data register
D5	Port G configuration register
D6	Port G input pins (read only)
D7	Port I input pins (read only)
D8 to DB	Reserved
DC	Port D data register
DD to DF	Reserved
E0 to EF	On-chip functions and registers
E0 to E8	Reserved
E9	MICROWIRE/PLUS shift register (SIOR)
EA	Timer lower byte
EB	Timer upper byte
EC	Timer autoload register lower byte
ED	Timer autoload register upper byte
EE	CNTRL control register
EF	PSW register
F0 to FF	16 on-chip RAM bytes mapped as registers
FC	X register
FD	SP register
FE	B register

10.11 RESET

The following initializations are performed by the COP880/840/820 at reset:

PORT D:	LOGIC HIGH
PORT G:	TRI-STATE
PORT L:	TRI-STATE
PC:	CLEARED
PSW and CNTRL:	CLEARED
B, X, SP:	UNKNOWN at power-on reset. UNCHANGED at external reset.
RAM:	UNKNOWN at power-on reset. UNCHANGED at external reset.
ACC and TIMER 1:	UNKNOWN at power-on reset. UNKNOWN at external reset with Crystal oscillator clock option selected. UNCHANGED at external reset with R/C or External oscillator clock options.
EECR:	CLEARED at power-on reset. The BsyERAM bit is set only by a write operation. This bit is cleared at the end of a write cycle regardless of the reset state. In other words, the BsyERAM bit is not cleared by an external reset which occurs in the middle of a write cycle. The BsyERAM bit will be cleared only when a write cycle is completed. In the initialization routine, the user should perform a dummy write to guarantee that this register is cleared after reset.

10.12 MASK OPTION(S)

The COP8620/8640 mask-selectable options are listed below. The options are programmed at the same time as the ROM pattern to provide the user with hardware flexibility.

Option 1: COP8620C/COP8640C CKI Input

=1 Crystal (CKI/10); CKO for crystal configuration

=2 External (CKI/10); CKO available as G7 input

=3 R/C (CKI/10); CKO available as G7 input

Option 2: COP8620C/8640C Bonding

=1	28-pin package
=2	N/A
=3	20-pin package

10.13 EMULATION DEVICES

The following chart shows the emulators available for the different COP8620/8640 packages. The emulators are discussed in detail in Appendix C.

Part Number	Emulator Package	Emulator (Type)
COP8620C-XXX/N COP8640C-XXX/N	28 DIP	COP8640CMHD (MCM[a])
COP8622C-XXX/N COP8642C-XXX/N	20 DIP	COP8642CMHD (MCM)
COP8620C-XXX/WM COP8640C-XXX/WM	28 LCC	COP8640CMEA[b] (MCM)
COP8622C-XXX/WM COP8642C-XXX/WM	20 SO	None

a. Multi-chip Module (UV Erasable)

b. Same footprint as 28-pin SO

Chapter 11

COP820CJ

11.1 INTRODUCTION

The COP820CJ is a member of the COPS 8-bit microcontroller family. It is a fully static microcontroller, fabricated using double-metal silicon gate microCMOS technology. This low-cost microcontroller is a complete microcomputer containing all system timing, interrupt logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include an 8-bit memory-mapped architecture, MICROWIRE serial I/O, a 16-bit timer/counter with capture register, and a multi-sourced interrupt. Each I/O pin has software selectable options to adapt the COP820CJ to specific applications. Several versions of the part are available that operate over different voltage and temperature ranges. Refer to the datasheet for more specfic information. High throughput is achieved with an efficient instruction set operating at a rate of 1 microsecond per instruction.

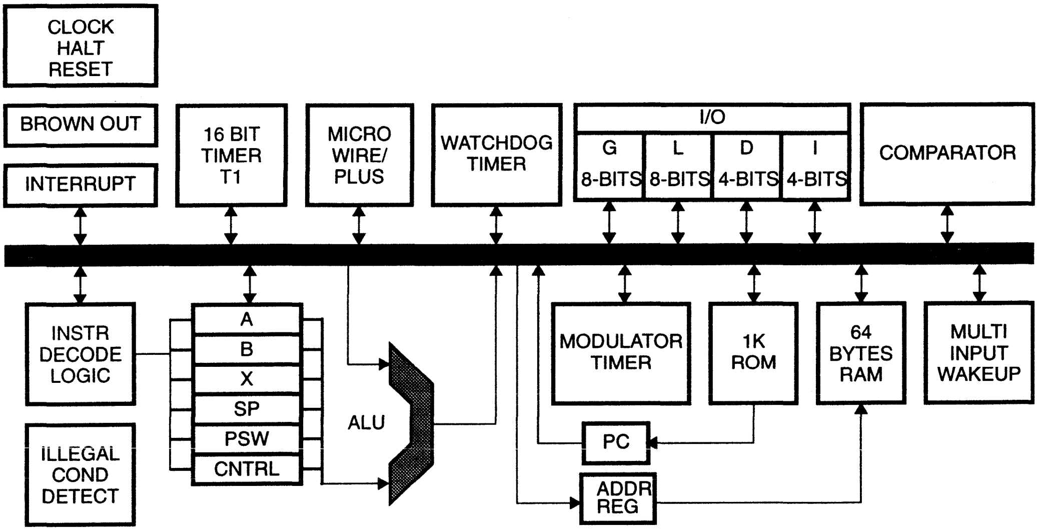
The COP820CJ has the following special features:

- Schmitt trigger inputs on Port L
- High current drive on pins L4 to L7
- Brown Out protection
- Watchdog Timer
- Modulator/Timer
- Comparator
- Multi-Input Wakeup on Port L
- 16-pin version in SO package

This chapter discusses the device specifics of the COP820CJ microcontroller. Information relevant to all COP800 Basic Family members is not covered in this chapter but may be found in the first eight chapters of this manual. In this chapter, the term “COP820CJ” refers to all COP820CJ packages, including the COP822CJ and COP823CJ.

11.2 BLOCK DIAGRAM

The diagram in Figure 11-1 shows the basic functional blocks associated with the COP820CJ. These blocks include the Arithmetic Logic Unit (ALU), Timer, MICROWIRE/PLUS, I/O ports, comparator, watchdog, modulator timer, Multi-Input Wakeup, Brown-Out circuit, and on-chip memory.



TL/DD/11208-1

Figure 11-1 COP820CJ Block Diagram

11.3 DEVICE PINOUT/PACKAGES

The COP820CJ is available in 16-pin SO, 20-pin DIP, 20-pin SO, 28-pin DIP, and 28-pin SO packages. Figure 11-2 shows the COP820CJ device package pinouts.

Refer to the COP820CJ datasheet for more information on the device packages.

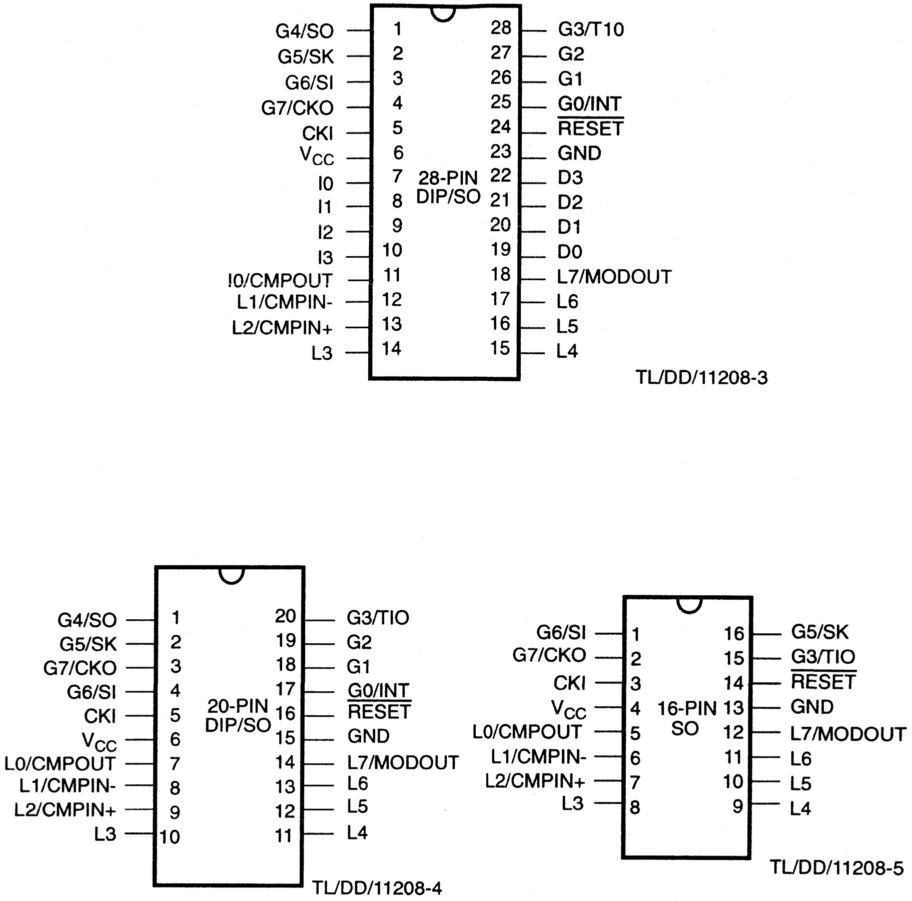


Figure 11-2 Device Package Pinouts

11.4 PIN DESCRIPTIONS

The COP820CJ has four dedicated function pins: V_{CC}, GND, CKI and $\overline{\text{RESET}}$. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. Table 11-1 lists the pin name, type, number and function of all COP820CJ signals.

Table 11-1 COP820CJ Pin Assignments

PORT	TYPE	ALTERNATE FUNCTION	16PIN SO	20 PIN DIP/SO	28 PIN DIP/SO
L0	I/O	MIWU/CMPOUT	5	7	11
L1	I/O	MIWU/CMPIN–	6	8	12
L2	I/O	MIWU/CMPIN+	7	9	13
L3	I/O	MIWU	8	10	14
L4	I/O	MIWU	9	11	15
L5	I/O	MIWU	10	12	16
L6	I/O	MIWU	11	13	17
L7	I/O	MIWU/MODOUT	12	14	18
G0	I/O	INTERRUPT		17	25
G1	I/O			18	26
G2	I/O			19	27
G3	I/O	TIO	15	20	28
G4	I/O	SO		1	1
G5	I/O	SK	16	2	2
G6	I	SI	1	3	3
G7	I/CKO	HALT RESTART	2	4	4
D0	O				19
D1	O				20
D2	O				21
D3	O				22
I0	I				7
I1	I				8
I2	I				9
I3	I				10
V_{CC}			4	6	6
GND			13	15	23
CKI			3	5	5
RESET			14	16	24

11.5 INPUT/OUTPUT PORTS

The number of I/O ports available on the COP820CJ device depends on package type. The COP820CJ 16- and 20-pin packages have only a Port L and Port G. The 28-pin COP820CJ parts have a Port L, Port G, Port I and Port D. All common COP800 ports are described in Chapter 7 of this manual. However, a brief description of each port is included in this section.

Port D, where available, is a 4-bit output-only port with moderately high sink current capability. The Port D pins have no alternate functions.

Port G is an 8-bit reconfigurable I/O port. Pins 0 - 5 of the port are configured by writing to the Port G configuration and data registers as described in Section 2.3.3. Pin G6 is a dedicated TRI-STATE input pin. Pin G7 is either an input or output, depending on the oscillator mask option selected. All Port G pins have Schmitt triggers on their inputs. The MICROWIRE/PLUS serial interface is implemented through pins G4, G5, and G6. Pin G4 is not available in the 16 pin package, limiting the 16-pin implementation to slave mode or just as a serial-shift input register. The Port G pins have the following alternate functions:

G0	INTR (External Interrupt Input)
G1	No alternate function
G2	No alternate function
G3	Timer 1 I/O
G4	S0 (MICROWIRE/PLUS Serial Data Output)
G5	SK (MICROWIRE/PLUS Clock I/O)
G6	SI (MICROWIRE/PLUS Serial Data Input)
G7	Dedicated CKO (Clock Output) with Crystal Oscillator Mask Option or HALT/ Restart (Exit HALT Mode) with RC or External Oscillator Mask Option

Port I, where available, is a 4-bit input-only port. All Port I pins are Hi-Z inputs. No alternate functions have been assigned to the Port I pins.

Port L is an 8-bit reconfigurable I/O port. The port is configured by writing to the Port L configuration and data registers as described in Section 2.3.3. Pins L4 to L7 have high sink current capability. Refer to the COP820CJ datasheet for specific Port L electrical characteristics. The Port L pins have the following alternate functions.

L0	MIWU or CMPOUT
L1	MIWU or CMPIN-
L2	MIWU or CMPIN+
L3	MIWU
L4	MIWU (high sink current capability)
L5	MIWU (high sink current capability)

L6	MIWU (high sink current capability)
L7	MIWU or MODOUT (high sink current capability)

The selection of alternate Port L functions is performed using registers WKEN (address 00C9 Hex) to enable MIWU, and CNTRL2 (address 00CC Hex) to enable the comparator and modulator. The programmer must always ensure that the Port L data and configuration registers are set to the correct values when using the alternate functions. For example, when using the comparator, the user must program the Port L pins used for the non-inverting and inverting terminals as inputs. Pin L0 must be programmed as an output if the output of the comparator is required on the pin. However, if there is an RC network on the inverting terminal which needs to be discharged as is the case in A/D conversion (see Chapter 13), pin L1 can be temporarily configured as an output set to logic 0 to discharge the capacitor, without having to change the entire comparator set-up.

Port L pins have Schmitt Triggers on their inputs. This reduces the noise sensitivity of the inputs, which is useful in applications working in electrically noisy environments such as industrial timers, appliances connected to the power line, and automotive applications.

11.6 PROGRAM MEMORY

The COP820CJ contains 1K bytes of program memory. All program memory for the COP820CJ devices is mask-programmed ROM. Refer to Chapter 2 for detailed information on the program memory.

11.7 DATA MEMORY

The COP820CJ has 64 bytes of RAM data memory. These 64 bytes are memory mapped into two different locations. The first 48 bytes are resident from address 0000 to 002F Hex, while the remaining 16 bytes (containing the register memory) are located from address 00F0 to 00FF Hex. Refer to Chapter 2 for information on the data memory architecture.

11.8 REGISTER BIT MAPS

The COP820CJ has five bit-mapped registers in addition to the PSW and CNTRL1 registers described in Section 2.4.3. The bit maps for these additional registers are shown below. PSW and CNTRL1 are also included for reference.

The WKEDG Register Bits are:

L0EDG	Pin L0 Wakeup Edge Select Bit
L1EDG	Pin L1 Wakeup Edge Select Bit
L2EDG	Pin L2 Wakeup Edge Select Bit
L3EDG	Pin L3 Wakeup Edge Select Bit

L4EDG	Pin L4 Wakeup Edge Select Bit
L5EDG	Pin L5 Wakeup Edge Select Bit
L6EDG	Pin L6 Wakeup Edge Select Bit
L7EDG	Pin L7 Wakeup Edge Select Bit

Table 11-2 WKEDG Register Bits (Address 00C8 Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
L7EDG	L6EDG	L5EDG	L4EDG	L3EDG	L2EDG	L1EDG	L0EDG

The WKEN Register Bits are:

L0EN	Pin L0 Wakeup Enable Bit
L1EN	Pin L1 Wakeup Enable Bit
L2EN	Pin L2 Wakeup Enable Bit
L3EN	Pin L3 Wakeup Enable Bit
L4EN	Pin L4 Wakeup Enable Bit
L5EN	Pin L5 Wakeup Enable Bit
L6EN	Pin L6 Wakeup Enable Bit
L7EN	Pin L7 Wakeup Enable Bit

Table 11-3 WKEN Register Bits (Address 00C9 Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
L7EN	L6EN	L5EN	L4EN	L3EN	L2EN	L1EN	L0EN

The WKPND Register Bits are:

L0PND	Pin L0 Wakeup Pending Bit
L1PND	Pin L1 Wakeup Pending Bit
L2PND	Pin L2 Wakeup Pending Bit
L3PND	Pin L3 Wakeup Pending Bit
L4PND	Pin L4 Wakeup Pending Bit
L5PND	Pin L5 Wakeup Pending Bit
L6PND	Pin L6 Wakeup Pending Bit
L7PND	Pin L7 Wakeup Pending Bit

Table 11-4 WKPND Register Bits (Address 00CA Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
L7PND	L6PND	L5PND	L4PND	L3PND	L2PND	L1PND	L0PND

The WDREG Register Bit is:

WDREN Watchdog Reset enable bit

Table 11-5 WDREG Register Bits (Address 00CD Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UNUSED	UNUSED	UNUSED	UNUSED	UNUSED	UNUSED	UNUSED	WDREN

The PSW Register Bits are:

GIE	Global interrupt enable (enables interrupts)
ENI	External interrupt enable
BUSY	MICROWIRE/PLUS busy shifting flag
IPND	External interrupt pending
ENTI	Timer 1 interrupt enable
TPND	Timer 1 interrupt pending (timer underflow or capture edge)
C	Carry Flip/Flop
HC	Half-Carry Flip/Flop

Table 11-6 PSW Register Bits (Address 00EF Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	TPND	ENTI	IPND	BUSY	ENI	GIE

The timer and MICROWIRE/PLUS control register bits are:

SL1 & SLO	Select the MICROWIRE/PLUS clock divide-by (00=2,01=4,1x=8)
IEDG	External interrupt edge polarity (0 = rising edge, 1 = falling edge)
MSEL	Selects G5 and G4 as MICROWIRE/PLUS signals SK and SO
TRUN	Used to start and stop the timer/counter (1 = run, 0 = stop)
TC1	Timer 1 Mode Control Bit
TC2	Timer 1 Mode Control Bit
TC3	Timer 1 Mode Control Bit

Table 11-7 CNTRL1 Register Bits (Address 00EE Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1	TC2	TC3	TRUN	MSEL	IEDG	SL1	SL0

The CNTRL2 Register Bits are:

MC3	Modulator/Timer Control Bit
MC2	Modulator/Timer Control Bit
MC1	Modulator/Timer Control Bit
CMPEN	Comparator Enable Bit
CMPRD	Comparator Read Bit
CMPOE	Comparator Output Enable Bit
WDUDF	Watchdog Timer Underflow Bit (Read Only)

Table 11-8 CNTRL2 Register Bits (Address 00CC Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MC3	MC2	MC1	CMPEN	CMPRD	CMPOE	WDUDF	UNUSED

11.9 MEMORY MAP

The COP820CJ is based on a memory mapped architecture. All data memory, I/O ports, port registers and function registers are mapped into the data memory address space. Table 11-9 shows the organization of the data memory address space and the mapping of specific addresses. Read-only memory locations are also noted in the table.

11.10 RESET

The COP820CJ has three types of reset: External Reset, Watchdog Reset, and Brown Out Reset. The following sections describe in detail the conditions required to generate these resets and the effect of each type of reset on the COP820CJ.

11.10.1 Reset Initialization

Table 11-10 shows the initialization performed by the COP820CJ during the External, Watchdog, and Brown Out resets. The effect of this initialization is to disable all interrupts, Timer 1, the Modulator/Timer, the Multi-Input Wakeup, the MICROWIRE/PLUS, and the Watchdog; to set all Port G and L pins to inputs; and to set Port D high.

Table 11-9 COP820CJ Memory Map

ADDRESS	CONTENTS
00 to 2F	On-chip RAM bytes
30 to 7F	Unused RAM address space (reads as all 1's)
80 to BF	Reserved (Reads Undefined Data)
C0 to C7	Reserved
C8	MIWU Edge Select Register (WKEDG)
C9	MIWU Enable Register (WKEN)
CA	MIWU Pending Register (WKPND)
CB	Reserved
CC	Control2 Register(CNTRL2)
CD	Watchdog Register(WDREG)
CE	Watchdog Counter (WDCNT)
CF	Modulator Reload (MODRL)
D0 to DF	On-chip I/O and registers
D0	Port L data register
D1	Port L configuration register
D2	Port L input pins (read only)
D3	Reserved
D4	Port G data register
D5	Port G configuration register
D6	Port G input pins (read only)
D7	Port I input pins (read only); upper 4 bits undefined
D8 to DB	Reseved for Port C
DC	Port D Data Register
DD to DF	Reserved for Port D
E0 to EF	On-chip functions and registers
E0 - E8	Reserved
E9	MICROWIRE/PLUS shift register (SIOR)
EA	Timer lower byte
EB	Timer upper byte
EC	Timer1 autoload register lower byte
ED	Timer1 autoload register upper byte
EE	CNTRL1 control register
EF	PSW register
F0 to FF	16 on-chip RAM bytes mapped as registers
FC	X register
FD	SP register
FE	B register

Table 11-10 Reset Initialization

Register/Port	Contents after External Reset	Contents after Watchdog Reset	Contents after Brown Out Reset
Port D	LOGIC HIGH	LOGIC HIGH	LOGIC HIGH
Port G	TRI-STATE	TRI-STATE	TRI-STATE
Port L	TRI-STATE	TRI-STATE	TRI-STATE
PC	CLEARED	CLEARED	CLEARED
RAM including SP, B and X	UNKNOWN at power-on UNAFFECTED with power already applied	UNKNOWN	UNKNOWN
Timer 1 and Accumulator	UNKNOWN at power-on UNKNOWN with power already applied (Crystal Oscillator option selected) UNAFFECTED with power already applied (R/C or External Oscillator option selected)	UNKNOWN	UNKNOWN
PSW CNTRL1	CLEARED CLEARED	CLEARED CLEARED	CLEARED CLEARED
CNTRL2	CLEARED	CLEARED	CLEARED except Bit1 (Bit1 is unaffected)
WKEDG WKEN WKPND	CLEARED CLEARED UNKNOWN	CLEARED CLEARED UNKNOWN	CLEARED CLEARED UNKNOWN
WDREG	CLEARED	CLEARED	CLEARED except Bit0 (Bit0 is unaffected)
Watchdog Prescaler	FF Hex normally ALTERED at external reset exit from HALT	FF Hex	FF Hex
Watchdog Counter	FF Hex normally ALTERED at external reset exit from HALT	FF Hex	FF Hex

NOTES: 1. Whenever V_{CC} is greater than the Brown Out voltage, the external reset has priority over the Brown Out reset. The external reset always has priority over the Watchdog reset.

2. During a Brown Out or External reset, the WDREN and WDUDF bits are cleared. However, if a Watchdog underflow caused the reset, the values of the WDREN and WDUDF bits are preserved, so that the initialization routine can detect whether a Watchdog underflow has occurred.

11.10.2 Reset Timing Considerations

For applications using the crystal or resonator clock option, a reset condition occurring during HALT mode automatically introduces a delay of 256 clock cycles immediately after the rising edge of an external $\overline{\text{RESET}}$ or the recovery of V_{CC} above the Brown Out voltage. This ensures that the crystal or resonator has had enough time to oscillate in a stable manner. The program starts execution at address 0000 Hex within 2 additional instruction cycles.

For all other external reset conditions, the program starts execution 2 cycles after the rising edge of the reset pulse.

The Brown Out reset initialization is started once V_{CC} has exceeded the Brown Out voltage. If an external reset takes place during this initialization, the external reset has priority over the Brown Out reset. The external reset always has priority over the Watchdog reset.

11.10.3 Power-On Reset Circuit

The external power-on reset circuit must normally meet the requirements for the COP800 Basic family reset circuit as described in Section 2.6. However, if the Brown Out protection option is used and the power supply rise time is greater than 50us, the standard power-on reset circuit should be omitted, saving three components. In this case, the $\overline{\text{RESET}}$ pin must be connected to V_{CC}. Power-on reset is then automatically performed by the Brown Out protection circuit.

11.10.4 Watchdog Reset

With Watchdog enabled, the Watchdog logic internally resets the device if the user program does not service the Watchdog timer within the selected service window. The Watchdog reset does not disable the Watchdog. Upon Watchdog reset, the Watchdog Prescaler/Counter are each initialized with 00FF Hex. The Watchdog reset does not have priority over any other COP820CJ resets. Refer to the Watchdog section of this chapter for more information on the Watchdog circuit.

11.10.5 Brown Out Reset

The on-board Brown Out protection circuit resets the device when the operating voltage (V_{CC}) goes below the Brown Out voltage. The device is held in reset when V_{CC} stays below the Brown Out voltage. The device comes out of reset as V_{CC} rises from a voltage lower than the Brown Out voltage and reaches the Brown Out voltage. If a two-pin crystal/resonator clock option is selected, the Brown Out reset will trigger a 256 t_c delay. This delay allows the oscillator to stabilize before the device exits the reset state. The delay is not used if the clock option is either R/C or external clock. The contents of data registers and RAM are unknown following a Brown Out reset. The external reset takes priority over Brown Out Reset and will deactivate the 256 t_c cycles delay if in progress. The Brown Out reset takes priority over the Watchdog reset.

11.10.6 External Reset

A COP820CJ master reset is generated by holding the external $\overline{\text{RESET}}$ pin low. This type of reset is common to all COP800 family devices, and is described in Section 2.6.

11.10.7 Reset Initialization Routine

The reset initialization routine cannot distinguish between an external reset and a Brown Out reset. Therefore, if the Brown Out detection option is used, the reset initialization routine must initialize the processor into a safe state, when used in safety-critical applications such as appliances connected to the power line. A watchdog underflow can be detected in the following way:

```
.=000
LD SP, #02F                    ; Initialize the stack pointer
IFBIT WDUDF,CNTRL2             ; Test whether reset is a Watchdog reset
JMP WDERR                      ; Execute the Watchdog error routine
...                            ; Normal initialization routine follows
```

11.11 BROWN OUT PROTECTION

The COP820CJ has an on-board Brown Out protection circuit for use in applications where temporary drops in the supply voltage (V_{CC}) could create potentially hazardous situations. For example, household appliances connected to the AC power line are subject to glitches or longer term drop-outs on the power supply, temporarily driving the microcontroller below its minimum operating voltage. Under such conditions, the program counter, stack and RAM contents are not guaranteed to be preserved. If this occurs, the program behavior is unpredictable and a potentially dangerous event may happen even when the power supply returns to its normal level.

The Brown Out protection circuit is permanently enabled or disabled via a mask option. If enabled, the Brown Out protection circuit monitors V_{CC} and compares it with the Brown Out voltage. If V_{CC} falls below the Brown Out voltage, the COP820CJ is held in reset. When V_{CC} rises above the Brown Out voltage, Brown Out initialization is generated as described in Section 11.8, ensuring that the program is restored to a defined state.

If Brown Out protection is disabled, the minimum operating voltage for the COP820CJ is 2.5V. If Brown Out protection is enabled, the minimum operating voltage is the Brown Out voltage, as long as the maximum operating frequency for the device at the Brown Out voltage is not exceeded. Refer to the datasheet for the dependency of minimum operating voltage on the maximum permissible operating frequency. The Brown Out voltage tracks the minimum operating voltage, so that devices with lower Brown Out voltages are guaranteed to operate at lower V_{CC} than devices with higher Brown Out voltages. Therefore, if Brown Out protection is enabled, the device is guaranteed to operate down to the Brown Out voltage even if this voltage is below 2.5V. For a temperature range of 0°C to 70°C the Brown Out voltage is expected to be between 1.9V and 3.9V. Over the full automotive temperature range, this extends from 1.8V to 4.2V.

If the device is intended to operate at a voltage lower than the maximum Brown Out voltage (VBO max), the Brown Out circuit should be disabled via the Brown Out mask option.

The Brown Out Circuit is active in HALT mode. This increases the HALT mode current by up to 100uA.

11.12 WATCHDOG

The COP820CJ has an on board 8-bit Watchdog timer. The timer contains an 8-bit READ/ WRITE down counter clocked by an 8-bit prescaler. The timer can be programmed to operate in either of two modes: Watchdog timer or a general-purpose counter. Figure 11-3 shows the Watchdog timer block diagram. The Watchdog counter is decremented every 256 t_C cycles.

Mode 1: Watchdog Timer

The Watchdog timer is intended for use in applications where glitches or other sources of external interference could corrupt the program counter or the stack contents. This can result in the program behavior being unpredictable and potentially dangerous. For example, the electrically noisy environment in which an automotive application is used could cause the application software to be stuck in an infinite loop or to execute look-up table data, thus causing unpredictable behavior. The programmer can already protect against execution of unused code by ensuring that these areas are filled with the value 00 Hex (software trap opcode), and by properly handling the software trap condition in the interrupt routine.

The Watchdog can be enabled or disabled only once after a Brown Out reset or External reset. On power-up, the Watchdog is disabled. If the Watchdog timer is enabled, the user program should write periodically into the 8-bit WDCNT counter, location 00CE Hex, before the counter underflows. The counter is loaded with N-1 to get N counts. The counter underflow causes a Watchdog reset as described in Section 11.8. Loading the 8-bit counter initializes the prescaler with FF Hex and starts the prescaler and the counter. Both the prescaler and the counter are stopped when the counter underflows. They are each loaded with FF Hex when the device goes into the HALT mode. The prescaler is used for crystal or resonator start-up when the device exits the HALT mode through Multi-Input Wakeup. In this case, the prescaler/counter contents are changed.

The programmer can adjust the required response time for a Watchdog failure by loading different values in the counter. If a very low value is loaded in WDCNT, the response time will be fast, but the counter must be updated more regularly. If a fast response time is not required, then a larger counter value will be sufficient.

Mode 2: Timer

In this mode, the prescaler/counter is used as a timer by keeping the WDREN (Watchdog reset enable) bit at zero. The counter underflow sets the WDUDF bit, but the underflow does not reset the device. Loading the 8-bit counter (load N-1 for N counts) sets the Watchdog Timer Enable (WDTEN) signal to "1", loads the prescaler with FF, and starts the timer. The counter underflow stops the timer.

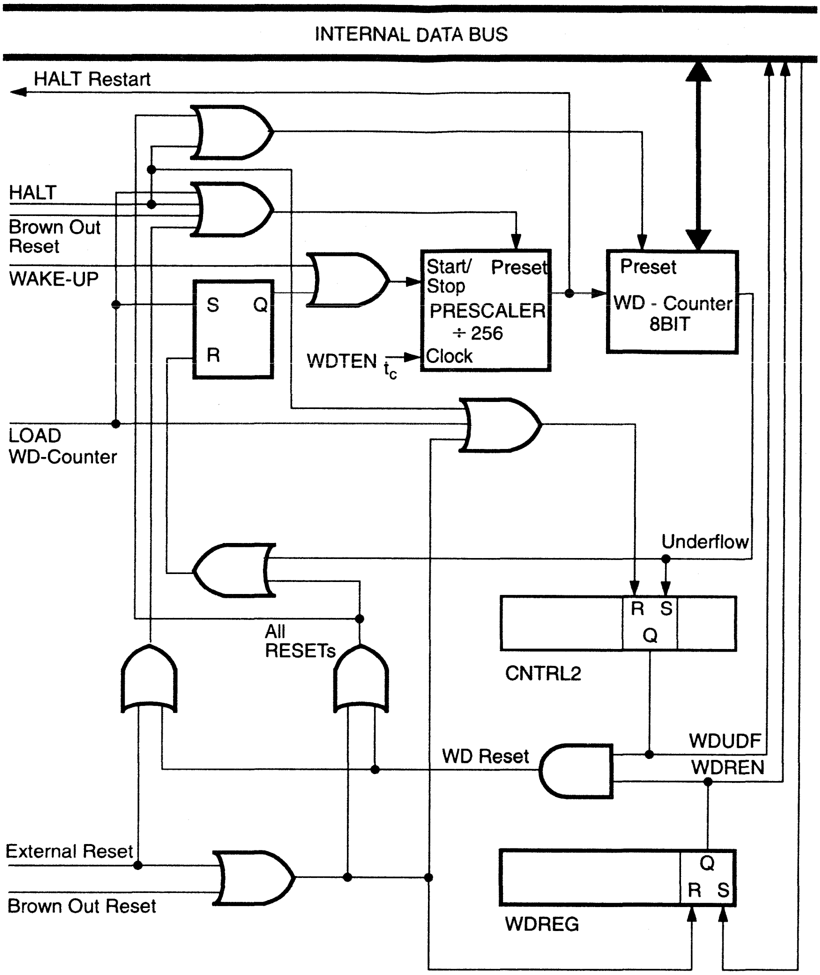


Figure 11-3 Watchdog Timer Block Diagram

The WDTEN signal serves as a start signal for the Watchdog timer. This signal is set to "1" when the 8-bit counter is loaded by the user program, either because of a Watchdog service or because of a write to the counter. The signal is set to "0" by a reset and is transparent to the user program.

Control and Status Bits

WDUDF: Watchdog Timer Underflow Bit

This bit resides in the CNTRL2 Register. The bit is set when the Watchdog timer underflows. The underflow resets the device if the Watchdog reset enable bit is set (WDREN=1). Otherwise, WDUDF can be used as the timer underflow flag. The bit is cleared upon Brown-Out reset, an External reset, a load to the 8-bit counter, or going into the HALT mode. It is a read-only bit.

WDREN: Watchdog Reset Enable Bit

This bit resides in Bit 0 of the WDREG register. This bit enables the Watchdog timer to generate a reset on a Watchdog timer underflow. The bit is cleared upon Brown Out reset or External reset. The bit is under software control but can be written to only once following a reset. After that, the hardware does not allow the bit to be changed during program execution. If WDREN= 1, Watchdog reset is enabled, otherwise it is disabled.

WDCNT: Watchdog Counter

This 8-bit read/write register contains the contents of the Watchdog timer. The contents can be read by the program at any time. The contents can be written at any time to update the Watchdog timer.

Initialization Example

The following code shows how to initialize the Watchdog:

```
.=000

...                       ; Reset initialization

LD WDCNT,#020             ; Load and start the timer

SBIT WDREN,WDREG          ; Enable the Watchdog reset

...                       ; Application code

LD WDCNT,#020             ; Regularly update the Watchdog
```

Table 11-11 shows the effect of Brown Out Reset, Watchdog Reset, and External Reset on the Control/Status bits.

Table 11-11 Effect of HALT, Reset and loading WDCNT on Watchdog Registers.

Parameter	HALT	Watchdog Reset	External or Brown Out Reset	Load Counter
8-bit Prescaler	FF Hex	FF Hex	FF Hex	FF Hex
8-bit WD Counter	FF Hex	FF Hex	FF Hex	User Value
WDREN bit	Unchanged	Unchanged	0	Unchanged
WDUDF bit	0	Unchanged	0	0
WDTEN Signal	Unchanged	0	0	1

11.13 MODULATOR/TIMER

The Modulator/Timer contains an 8-bit counter which is not memory mapped, and an 8-bit autoreload register, MODRL (address 00CF Hex). The Modulator/Timer has three modes of operation selected by the Modulator/Timer Control bits. These bits, MC1, MC2 and MC3 reside in the CNTRL2 Register.

Mode 1: MODULATOR

The Modulator mode is used to generate bursts of between 1 and 256 pulses at a frequency of either CKI or CKI/10. Figure 11-4 illustrates the timer configuration and the waveform generated by this mode. This type of waveform is used in remote control applications, such as electronic keys in the automotive area or remote control units for consumer appliances. Refer to the COP820CJ datasheet for the specification on the maximum permissible CKI frequency for the Modulator output.

The Modulator mode is selected by setting MC3 = 1. MC2 selects the modulator input clock. If MC2 = 1, the modulator input clock is set to CKI. If MC2 = 0, the modulator input clock is set to t_C. MC1 is used as the start bit for the modulator. The high frequency pulses are generated on the modulator output pin L7, which should be configured as an output, by setting bit 7 of register PORTLC. The number of pulses is determined by the 8-bit autoreload register MODRL, which is loaded with N-1 to get N pulses. Loading MODRL with FF Hex gives the maximum number of counts, 256. The user loads MODRL with the desired number of counts and sets MC1 to start the counter. MODRL is then loaded into the counter, and pulses at the modulator input frequency are routed to pin L7 until the counter underflows. On underflow the hardware resets MC1 and stops the counter. The L7 pin goes low and stays low until the counter is restarted by the user program. Unless the number of counts is changed, the user program does not have to load MODRL each time the counter is started. The counter can be started simply by setting the MC1 bit. Setting MC1 by software will load the counter with the value of the autoreload register. The software can reset MC1 to stop the counter.

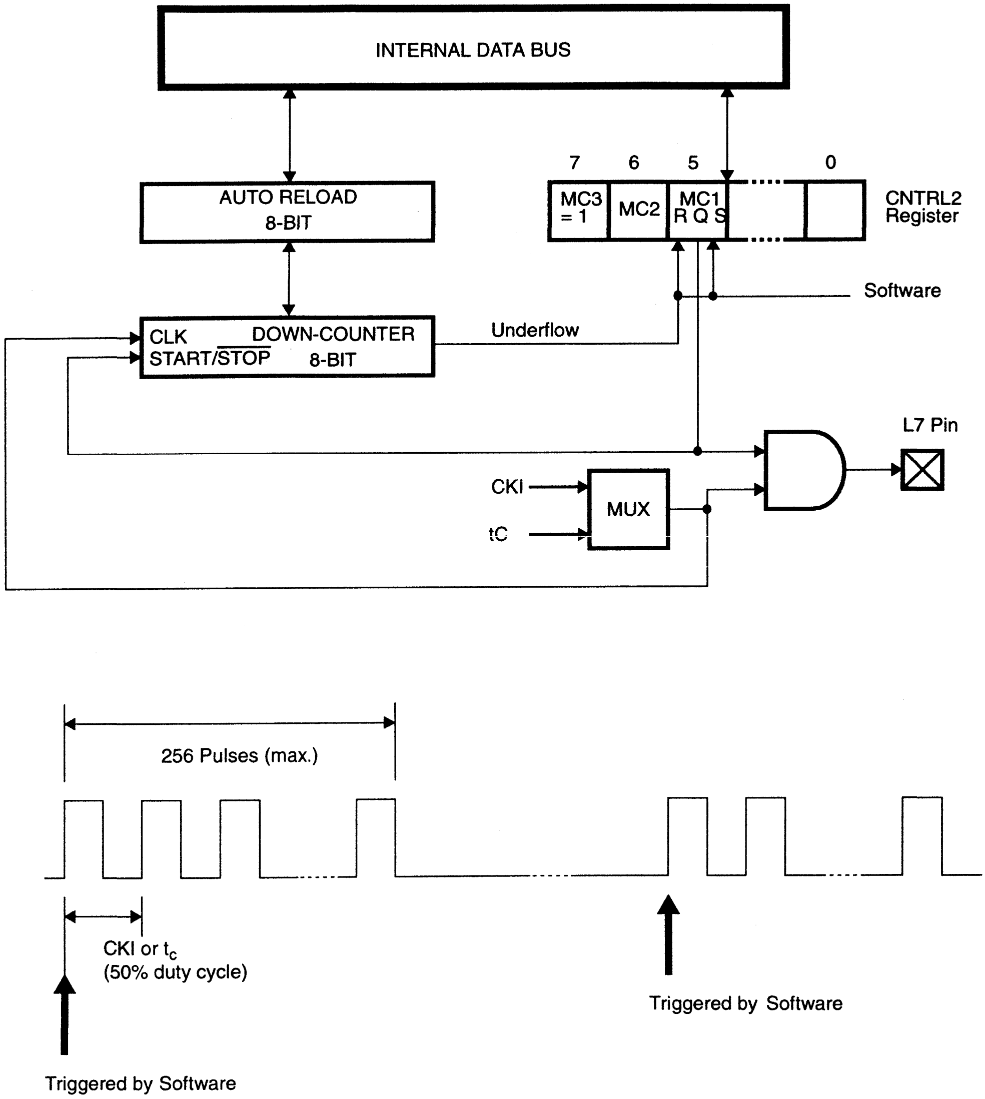
Example: Produce 10 pulses at a frequency of 2 MHz on L7

Use a 2 MHz crystal oscillator.

```
RBIT 7,PORTLD          ; Pin L7 output logic 0
SBIT 7,PORTLC          ;
SBIT MC3,CNTRL2        ; Choose modulator mode
SBIT MC2,CNTRL2        ; Choose CKI clocking
LD MODRL,#9            ; Load with 9 to get 10 pulses
SBIT MC1,CNTRL2        ; Start the modulator
```

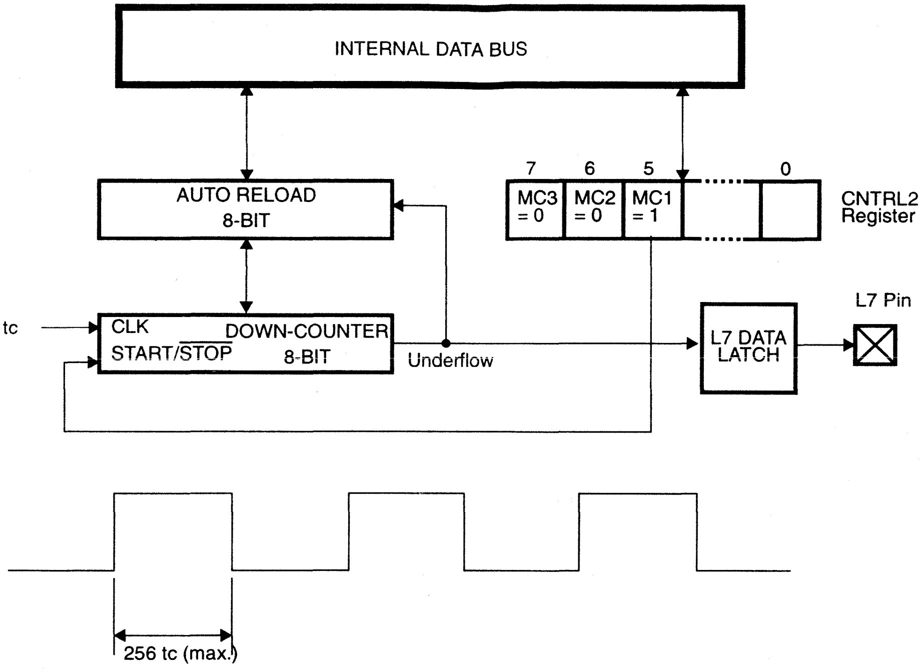
Mode 2: PWM TIMER, 50% duty cycle

The 50% duty cycle mode is used to generate a square wave without processor intervention. Applications include household appliances such as irons and coffee-makers that require a simple buzzer function. Timer 1 is then available for A/D conversions. Figure 11-5 illustrates the timer configuration and the waveform generated by this mode.



COP800-17

Figure 11-4 Modulator Block Diagram/Output Waveform



COP800-18

Figure 11-5 Mode 2: 50% Duty Cycle Output

If both MC2 and MC3 are 0, a 50% duty cycle signal is generated on pin L7. This pin must be configured as an output pin. In this mode the 8-bit counter is clocked by t_C. The user loads MODRL with the desired number of counts. Loading MODRL with N-1 will give an output "on" time of N x t_C or a frequency of 1/(2N x t_C). Setting the MC1 control bit by software loads the counter with the value of the autoreload register and starts the counter. The counter underflow toggles the L7 output pin, thereby generating a waveform with a 50% duty cycle. The software can reset MC1 to stop the counter.

Example: Production of a 2kHz tone on L7

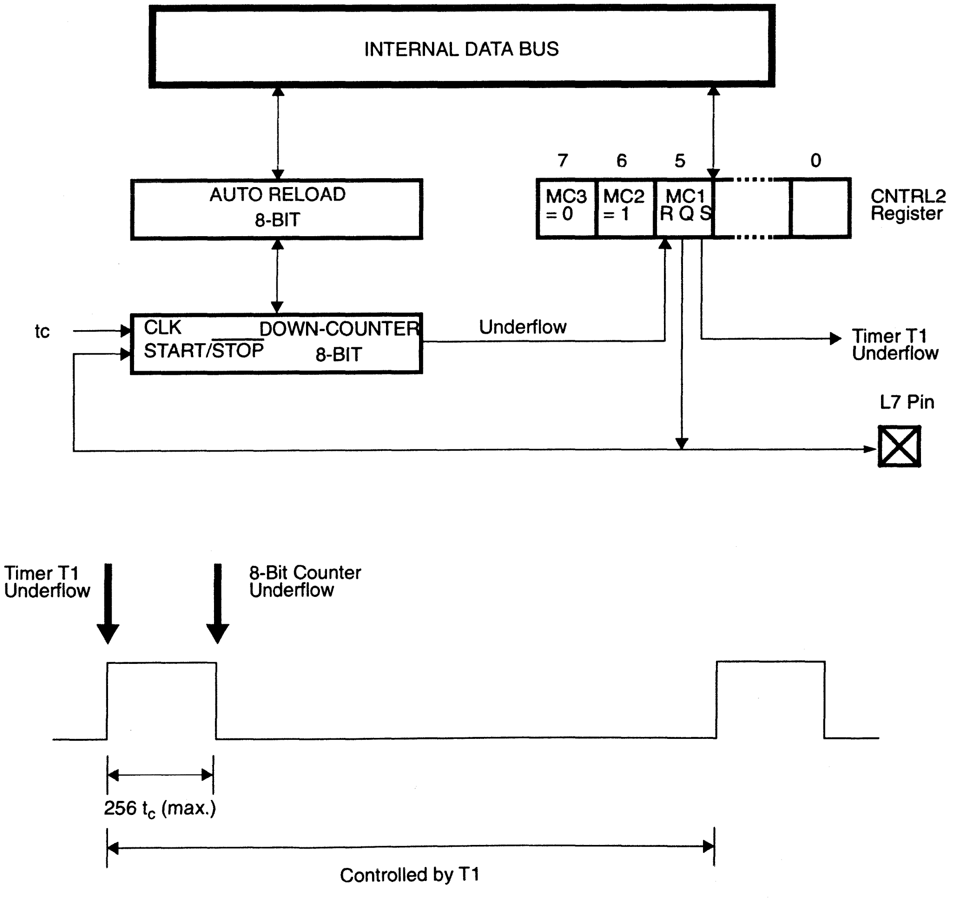
Using a 2 MHz crystal, t_C is 5us. The period of a 2 kHz tone is 500us. The "on" time is half of this, which is 250us. The value 250/5 = 50 should be loaded in MODRL.

```
RBIT 7,PORTLD            ; Pin L7 output logic 0
SBIT 7,PORTLC            ;
RBIT MC2,CNTRL2          ; Choose 50% duty cycle mode
RBIT MC3,CNTRL2          ;
LD MODRL,#49             ; Load with 49 to get 50 tC = 250 us period
SBIT MC1,CNTRL2          ; Start the tone
```

Mode 3: PWM TIMER, variable duty cycle

The variable duty cycle mode is used to generate two distinct types of waveforms without processor intervention. Figure 11-6 illustrates the timer configuration and the waveform generated by this mode.The waveform "on" time is determined by MODRL and the period is determined by the Timer 1 underflow. If the Timer 1 underflow occurs every 256 t_C cycles, an 8-bit PWM signal is generated, suitable for conversion into an analog signal. This type of signal is useful in DC motor control. If the underflow occurs every 512, 1024, 2048 etc. t_C cycles, 8-bit resolution over a voltage range of 0–0.5V_{CC}, 0–0.25V_{CC}, 0–0.125V_{CC}, etc. is possible. Delayed pulse generation is also possible with this mode. In this case, the Timer 1 register is loaded with the delay time and MODRL with the pulse width. This technique is useful for the phase control of AC-driven loads in electric drills, food mixers, washing machines and vacuum cleaners.

When MC3 = 0 and MC2 = 1, a variable duty cycle PWM signal is generated on the L7 pin, which should be configured as an output. The counter is clocked by t_C. In this mode the 16-bit Timer 1, along with the 8-bit down counter, are used to generate a variable



COP800-19

Figure 11-6 Mode 3: Variable Duty Cycle Output

duty cycle PWM signal. The programmer resets the Timer 1 start bit (bit 4 of CNTRL) during initialization. Then, the Timer 1 register is loaded with the required delay value in t_C cycles, the Timer 1 reload register is loaded with the desired repetition rate, and MODRL is loaded with the desired pulse width. In each case, loading the register with N-1 will give a time of N. Timer 1 must be configured in "PWM Mode/Toggle TIO Out" (CNTRL Bits 7,6,5 = 101). Setting bit 4 of CNTRL starts the sequence.

Each timer Timer 1 underflow sets MC1, which in turn loads the down counter with MODRL, starts the 8-bit counter, and sets L7 high. When the counter underflows, the MC1 control bit is reset and the L7 output goes low until the next timer Timer 1 underflow.

Table 11-12 shows the different operation modes for the Modulator/Timer.

Table 11-12 Modes of PWM timer

Control bits in CNTRL2			**Operation Mode L7 Function**
MC3	**MC2**	**MC2**	
0	0	0	Normal I/O
0	0	1	50% duty cycle mode (clocked by t_c)
0	1	X	Variable duty cycle mode (clocked by t_c) using Timer 1 underflow
1	0	X	Modulator mode (clocked by t_c)
1	1	X	Modulator mode (clocked by CKI)
NOTE:	MC1, MC2 and MC3 control bits are cleared upon reset.		

11.14 COMPARATOR

The COP820CJ has one differential comparator. Consumer appliances that must measure temperature or pressure can use this comparator to perform analog to digital conversion. Automotive applications that drive motors need to determine whether the starter motor is turning, because this reduces the battery voltage so much that the electric motor may stall. The comparator can test for this condition.

Pins L0, L1 and L2 are used for the comparator. The output of the comparator can be connected to the L0 pin, read by software, or both. The pins are assigned as follows:

L0	Comparator output
L1	Comparator inverting input
L2	Comparator non-inverting input

Comparator Control and Status Bits

These bits reside in register CNTRL2 (Address 00CC Hex):

CMPEN	Enables comparator ("1" = enable, "0" = disable)
CMPRD	Reads comparator output internally (CMPEN = 1, CMPOE = X)
CMPOE	Enables comparator output to pin L0 ("1" = enable, "0" = disable)

To enable the comparator, the programmer should set up L1 and L2 as high impedance inputs using PORTLD and PORTLC, and set the CMPEN bit. The comparator output can be viewed by reading the CMPRD bit. To enable the comparator output, set up L0 as an output using PORTLC, and set the CMPOE bit. If CMPOE is cleared, CMPEN is set and pin L0 is configured as an output, pin L0 will be set to 0V.

The comparator Select/Control bits are cleared after a reset, disabling the comparator. To save power, the program should disable the comparator before the device enters HALT mode.

The comparator rise and fall times are symmetrical. Refer to the COP820CJ datasheet for information on the DC and AC characteristics of the comparator.

A programming example for the comparator is given in the Applications chapter showing an A/D conversion with the COP820CJ.

11.15 MULTI-INPUT WAKEUP

The Multi-Input Wakeup feature is used to wake up the device from the HALT mode by means of a transition on one or more of the Port L pins. This feature is useful when the microcontroller has to exit the HALT mode from more than one external wakeup condition. For example, a remote control unit with fifty keys must exit HALT mode as soon as any one of the many keys is pressed. This can be implemented on the COP820CJ by arranging the keys in a matrix and using Multi-Input Wakeup on the key matrix inputs.

Figure 11-7 shows the block diagram for the Multi-Input Wakeup feature. This feature can also be used for latching high-to-low or low-to-high transitions occurring on the selected Port L pins. This does not require the use of HALT mode.

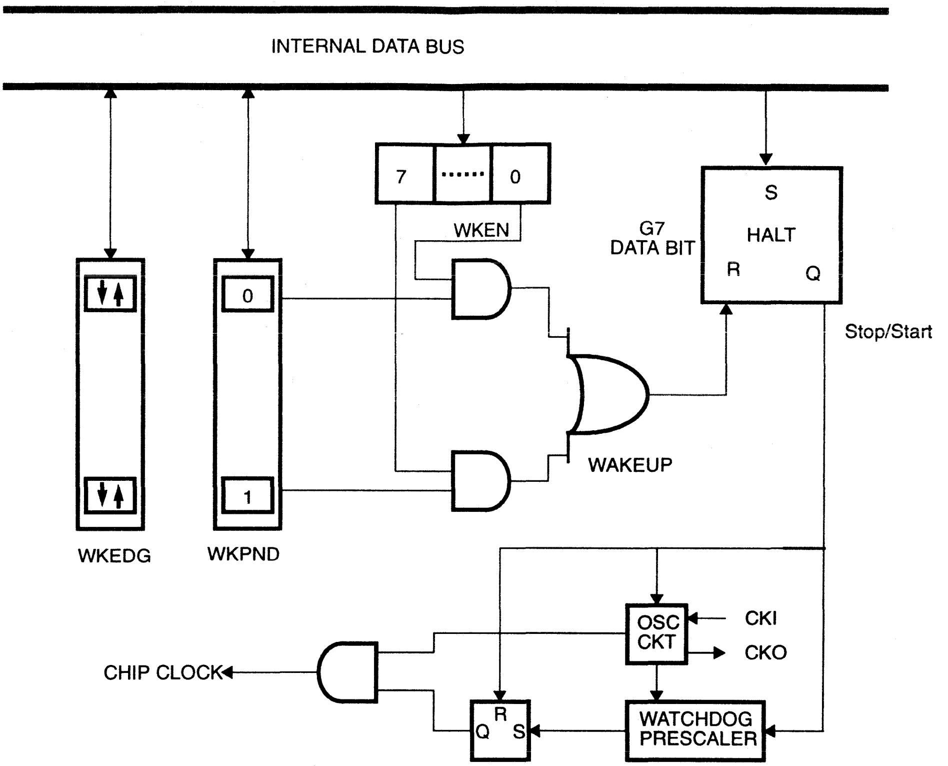
Multi-Input Wakeup Registers

WKEN Contains bits to enable Multi-Input Wakeup for individual Port L pins ("1" = enabled)

WKEDG Contains bits to select the type of transition sensed on individual Port L pins ("1" = negative edge)

WKPND Contains Wakeup Pending flags for individual Port L pins ("1" = pending)

When using this feature, the programmer must configure the Port L pins intended for use as Wakeup signals as inputs and clear the WKPND register. The programmer should program the corresponding bits of the WKEDG register. To enable Wakeup on a rising



COP800-20

Figure 11-7 Multi-Input Wakeup Logic

edge for a particular pin, the programmer should reset the associated WKEDG bit to 0. To enable Wakeup on a falling edge for a particular pin, the programmer should set the associated WKEDG bit to 1. Finally, the programmer should select which particular Port L bit or combination of Port L bits will be configured as Multi-Input Wakeup pins, by setting the respective bits in WKEN.

As soon as any one or more of the WKEN bits have been set, the occurrence of any one or more of the selected trigger conditions on the relevant Port L pins causes the associated bits in WKPND to be set to 1. If any bit of the WKPND register is set while the COP820CJ is in HALT mode, the part will exit the HALT mode within two t_C cycles if the External or RC clock options have been used, or after an additional delay of 256 t_C cycles if the Crystal/Resonator option has been used. This 256 t_C delay is generated by using the Watchdog prescaler. If the program attempts to enter HALT mode while any bit is still set in the WKPND register and its associated bit in the WKEN register is also set, the device will not enter HALT mode. It is the responsibility of the programmer to ensure that the WKPND register is cleared before attempting to enter HALT mode.

An Application Using Multi-Input Wakeup

Figure 11-8 shows the circuit diagram for a battery-powered remote control unit. The function of the unit is to transmit a specific code, whenever a particular key is pressed, using an infra-red LED driven from the modulator output L7. The additional LED connected to pin G3 is turned on by the software whenever a key is pressed to indicate to the user that the unit is functioning correctly. In order to conserve battery power, the unit is held in HALT mode until a key is pressed. The COP820CJ should come out of HALT mode whenever any key is pressed. The key is then decoded, the relevant code transmitted, and the part set back into HALT mode.

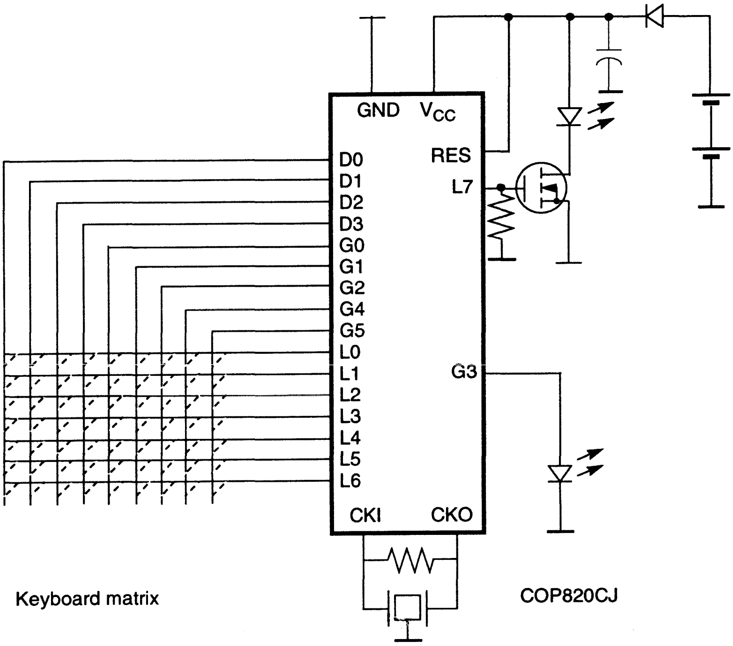


Figure 11-8 Battery-Powered Remote Control Unit

The Multi-Input Wakeup feature may be used to meet these requirements. After the end of the previous transmission, the keyboard is initialized by using the following procedure. The program sets up pins L0-L6 as inputs with weak pull-up resistors and Port D to logic 0. Pins G0, G1, G2, G4 and G5 are set by the software to output logic 0. The program sets bits 0-6 of the WKEDG register to one, indicating that the COP820CJ will wake up after a high-to-low transition on these pins. The WKPND register is cleared and then WKEN is loaded with 07F hex by the program, enabling Wakeup on pins L0-L6 and disabling Wakeup on pin L7. Finally, the program is put in HALT mode.

As soon as any one of the keys is pressed, the Port L pin in the same row as the key is driven down from its weak pull-up high state to zero, causing a high-to-low transition. This generates a Wakeup signal. Because the crystal oscillator option has been selected, the COP820CJ waits for 256 t_C cycles before proceeding.

The program now interrogates the WKPND register to determine the row of the key that has been pressed, and continues with keyboard scanning, debouncing, decoding, and transmission routines.

Example of Keyboard Initialization Routine

```
 ...
 LD WKEN,#000          ; Suspend Wakeup feature during setup
 LD PORTLD,#07F        ; Pins L0-L6 weak pull-ups, pin L7 output logic low
 LD PORTLC,#080        ;
 LD PORTD,#000         ; Pins D0-D3 logic low
 LD PORTGD,#000        ; Pins G0-G5 outputs logic low
 LD PORTGC,#03F        ;
 LD WKEDG,#07F         ; L0-L6 active on high-to-low transition
 LD WKPND,#000         ; Clear Wakeup pending flags
 LD WKEN,#07F          ; Enable Wakeup on L0-L6
 SBIT 7,PORTGD         ; Enter HALT mode
 NOP
 NOP
 ...                   ; After HALT exited, WKPND scanned to identify row
```

11.16 MASK OPTIONS

The COP820CJ mask-selectable options are listed below. The options are programmed at the same time as the ROM pattern to provide the user with hardware flexibility.

Option 1: CKI Input

=1 Normal Mode Crystal (CKI/10); CKO for crystal configuration

=2 Normal Mode External (CKI/10); CKO available as G7 input

=3 R/C (CKI/10); CKO available as G7 input

Option 2: Brown Out

=1 Enable Brown Out Protection (increased HALT current)

=2 Disable Brown Out Protection

Option 3: Bonding

=1 28-pin DIP

=2 20-pin DIP/SO

=3 16-pin SO

=4 28-pin SO

11.17 EMULATION DEVICES

The following chart shows the emulators available for the different COP820CJ packages. The emulators are discussed in detail in Appendix C.

Part Number	Emulator Package	Emulator (Type)
COP820CJ-XXX/N	28 DIP	COP820CJMHD (MCM[a])
COP820CJ-XXX/WM	28 SO	COP820CJMHEA[b] (MCM)
COP822CJ-XXX/N	20 DIP	COP822CMHD (MCM)
COP822CJ-XXX/WM	20 SO	NONE
COP823CJ-XXX/WM	16 SO	NONE

[a] Multi-chip Module (UV Erasable)

[b] Same footprint as 28-pin SO

Chapter 12

COP8780C

12.1 INTRODUCTION

The COP8780C is a member of the COPS microcontroller family. It is a fully static part, fabricated using double-metal, double poly silicon gate microCMOS EPROM technology. This device is available as UV erasable or One Time Programmable (OTP). This low-cost microcontroller is a complete microcomputer containing all system timing, interrupt logic, EPROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include an 8-bit memory-mapped architecture, MICROWIRE/PLUS serial I/O, a 16-bit timer/counter with capture register and a multi-sourced interrupt. Each I/O pin has software selectable options to adapt the COP8780C to specific applications. High throughput is achieved with an efficient instruction set operating at a rate of 1 microsecond per instruction.

This chapter discusses the device specifics of the COP8780C microcontroller. Information relevant to all COP800 Basic Family members is not covered in this chapter, but may be found in the first eight chapters of this manual. In this chapter, the term "COP8780" refers to all COP8780C packages, including the COP8781C and COP8782C.

12.2 BLOCK DIAGRAM

The diagram in Figure 12-1 shows the basic functional blocks associated with the COP8780. These blocks include the Arithmetic Logic Unit (ALU), Timer, MICROWIRE/PLUS, I/O ports, and on-chip memory.

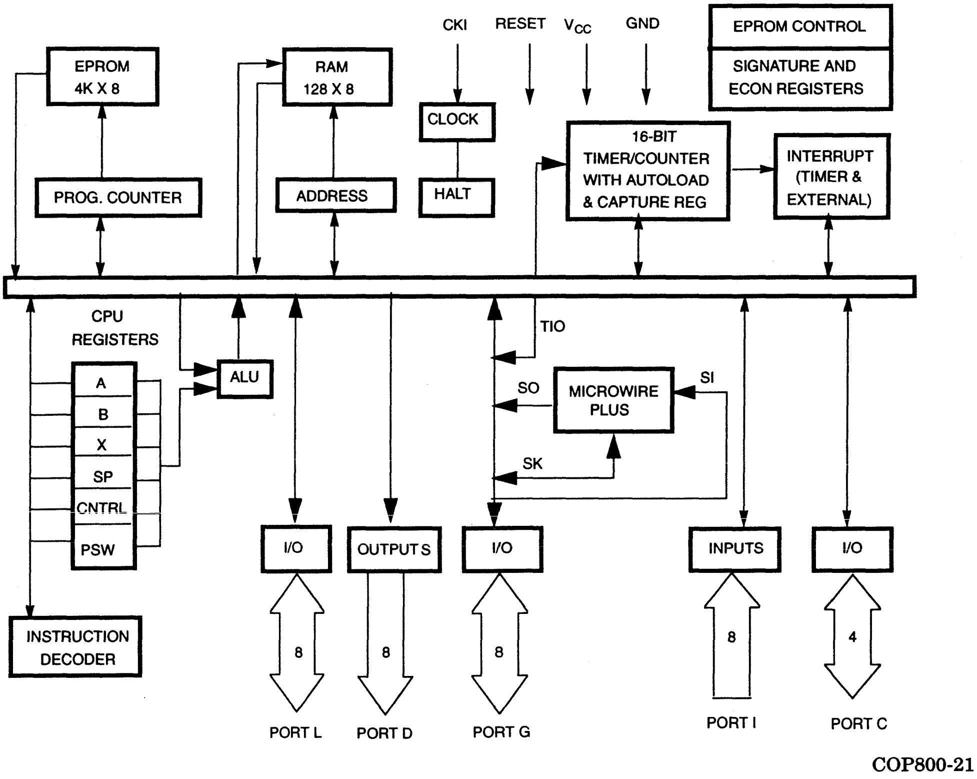


Figure 12-1 COP8780 Block Diagram

12.3 DEVICE PINOUT/PACKAGES

The COP8780 is available in UV erasable or OTP 20-pin DIP/SO, 28-pin DIP/SO, 40-pin DIP and 44-pin PLCC packages. Figure 12-2 shows the COP8780 device package pinouts.

Refer to the COP8780 datasheets for more information on the device packages.

12.4 PIN DESCRIPTIONS

The COP8780 has four dedicated function pins: V_{CC}, GND, CKI and $\overline{\text{RESET}}$. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. Table 12-1 lists the pin name, type, number and function of all COP8780 signals.

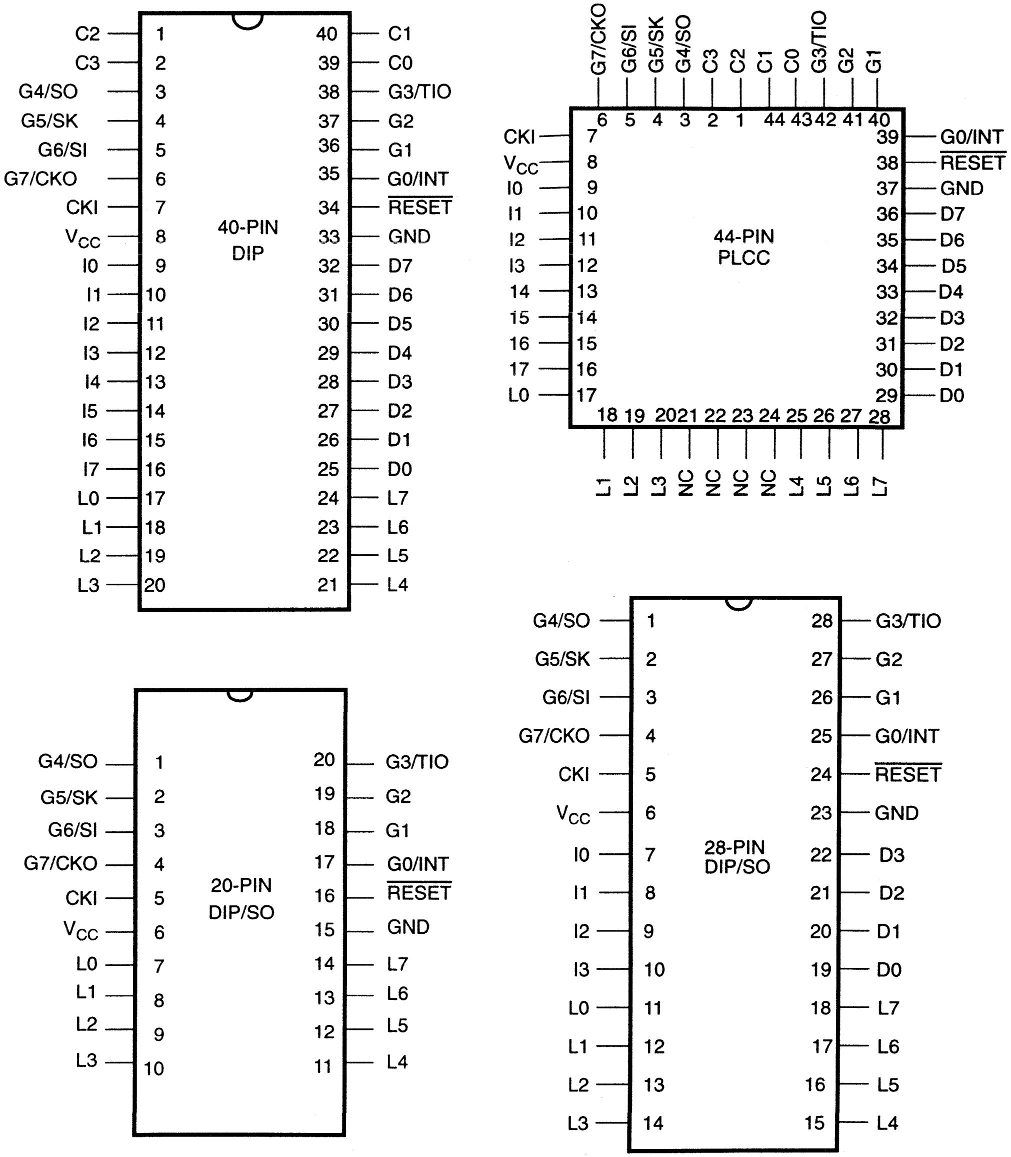


Figure 12-2 Device Package Pinouts

Table 12-1 COP8780 Pin Assignments

PORT	TYPE	ALTERNATE FUNCTION	20 PIN DIP/SO	28 PIN DIP/SO	40 PIN DIP	44 PIN PLCC
L0	I/O		7	11	17	17
L1	I/O		8	12	18	18
L2	I/O		9	13	19	19
L3	I/O		10	14	20	20
L4	I/O		11	15	21	25
L5	I/O		12	16	22	26
L6	I/O		13	17	23	27
L7	I/O		14	18	24	28
G0	I/O	INTERRUPT	17	25	35	39
G1	I/O		18	26	36	40
G2	I/O		19	27	37	41
G3	I/O	TIO	20	28	38	42
G4	I/O	SO	1	1	3	3
G5	I/O	SK	2	2	4	4
G6	I	SI	3	3	5	5
G7	I/CKO	HALT RESTART	4	4	6	6
D0	O			19	25	29
D1	O			20	26	30
D2	O			21	27	31
D3	O			22	28	32
I0	I			7	9	9
I1	I			8	10	10
I2	I			9	11	11
I3	I			10	12	12
I4	I				13	13
I5	I				14	14
I6	I				15	15
I7	I				16	16
D4	O				29	33
D5	O				30	34
D6	O				31	35
D7	O				32	36
C0	I/O				39	43
C1	I/O				40	44
C2	I/O				1	1
C3	I/O				2	2
V_{CC}			6	6	8	8
GND			15	23	33	37
CKI			5	5	7	7
RESET			16	24	34	38

12.5 INPUT/OUTPUT PORTS

The number of I/O ports available on the COP8780 device depends on package type. The COP8780 20-pin packages have only a Port L and Port G. The 28-pin COP8780 parts have a Port L, Port G, Port I and Port D. The 40- and 44-pin COP8780 packages have a Port C in addition to the ports available on the 28-pin packages. All common COP800 ports are described in Chapter 7 of this manual. However, a brief description of each port is included in this section.

Port C, where available (40 pin DIP and 44 pin PLCC packages), is a 4-bit reconfigurable I/O port. The port is configured by writing to the Port C configuration and data registers as described in Section 2.5.3. Reading bits 4 - 7 of the Port C registers and input pins returns undefined data. It is the user's responsibility to mask out the upper four bits when reading the Port C. This is accomplished by simply ANDing the Port C data with the value 000F Hex. This will ensure that the upper four bits of the Port C data are cleared. The Port C pins have not been assigned alternate functions.

Port D, where available, is a 4-bit (28 pin DIP/SO) or 8-bit (40 pin DIP and 44 pin PLCC) output only port. When writing an 8-bit quantity to devices which only have a 4-bit D Port, only the lower four bits are used. The Port D pins have no alternate functions.

Port G is an 8-bit reconfigurable I/O port. Pins 0-5 of the port are configured by writing to the Port G configuration and data registers as described in Section 2.5.3. Pin G6 is a dedicated input pin. Pin G7 is either an input or output, depending on the oscillator option selected. The Port G pins have the following alternate functions:

G0 INTR (External Interrupt Input)

G1 No alternate function

G2 No alternate function

G3 Timer 1 I/O

G4 S0 (MICROWIRE/PLUS Serial Data Output)

G5 SK (MICROWIRE/PLUS Clock I/O)

G6 SI (MICROWIRE/PLUS Serial Data Input)

G7 Dedicated CKO (Clock Output) with Crystal Oscillator Mask Option or HALT/Restart (Exit HALT Mode) with RC or External Oscillator Mask Option

Port I, where available, is a 4-bit (28 pin DIP/SO) or 8-bit (40 pin DIP and 44 pin PLCC) input-only port. All Port I pins are Hi-Z inputs. On the devices which only have a 4-bit I port, reading bits 4 - 7 of Port I will return undefined data. The user should mask out the upper four bits on these devices. No alternate functions have been assigned to the Port I pins.

Port L is an 8-bit reconfigurable I/O port. The port is configured by writing to the Port L configuration and data registers as described in Section 2.5.3. The Port L pins have no alternate functions.

12.6 PROGRAM MEMORY

The COP8780 contains 4096 bytes of UV-erasable or OTP EPROM memory. This memory is mapped in the program memory address space from 0000 to 0FFF Hex. The program memory may contain either instructions or data constants, and is addressed by the 15-bit program counter (PC). The program memory can be indirectly read by the LAID (Load Accumulator Indirect) instruction for table lookup of constant data. Program memory is discussed in detail in Chapter 2.

All locations in the EPROM program memory contain FF Hex (all 1's) after the COP8780 is erased. OTP parts are shipped with all locations already erased to FF Hex. Unused EPROM locations should always be programmed to 00 Hex so that the software trap can be used to halt runaway program operation.

The COP8780 can be configured to inhibit external reads of the program memory. This is accomplished by programming the security bit in the ECON (EPROM configuration) register to zero. See Section 12.8 for details.

12.7 DATA MEMORY

The data memory address space on the COP8780 includes on-chip RAM, I/O, and registers. The COP8780 can be configured to have either 64 or 128 bytes of RAM, depending on the value of the "RAM SIZE" bit in the ECON register. If the 64-byte RAM option has been selected, then the first 48 bytes are mapped from locations 0000 Hex through 002F Hex. The remaining 16 bytes are mapped from locations 00F0 Hex through 00FF Hex. If the 128-byte RAM option has been selected, then the first 112 bytes are mapped from locations 0000 Hex through 006F Hex. Again, the remaining 16 bytes are mapped from locations 00F0 Hex through 00FF Hex. Refer to Chapter 2 for details on the data memory architecture.

12.8 ECON (EPROM CONFIGURATION) REGISTER

The ECON register is used to configure the user-selectable clock, security, and RAM size options. The register can be programmed and read only in EPROM programming mode. Therefore, the register should be programmed at the same time as the program memory locations 0000 through 0FFF Hex. UV-erasable parts are shipped with FF Hex in this register. OTP parts are shipped with 7F Hex in this register.

The COP8780C has a security feature which prevents reading of the EPROM program memory when enabled. The security bit in the ECON register determines whether security is enabled or disabled. If the security option is enabled, then any attempt to externally read the contents of the EPROM results in the value 00E0 Hex being read from all program memory locations. If the security option is disabled, the contents of the internal EPROM may be read. The ECON register is readable regardless of the state of the security bit.

The format of the COP8780 ECON register is as follows:

Table 12-2 ECON Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	X	SECURITY	CKI 2	CKI 1	X	RAM SIZE	X

Bit 7	=	0	Must be programmed to zero.
Bit 6	=	X	Don't care.
Bit 5	=	1	Security disabled. External read and write access are allowed to EPROM.
	=	0	Security enabled. External read and write access are not allowed to EPROM.
Bits 4,3	=	1,1	External CKI option selected.
	=	0,1	Not allowed.
	=	1,0	RC oscillator option selected.
	=	0,0	Crystal oscillator option selected.
Bit 2	=	X	Don't care.
Bit 1	=	1	Selects 128 byte RAM option. This emulates COP840 and COP880.
	=	0	Selects 64 byte RAM option. This emulates COP820.
Bit 0	=	X	Don't care.

12.9 REGISTER BIT MAPS

The COP8780 devices have two registers that contain hardware control flags and bits. These registers, CNTRL and PSW, are located in the COP800 core and are described in the CORE REGISTERS section of this manual. The bit maps for these registers are shown below.

The PSW register bits are:

GIE	Global interrupt enable (enables interrupts)
ENI	External interrupt enable
BUSY	MICROWIRE/PLUS busy shifting flag
IPND	External interrupt pending
ENTI	Timer 1 interrupt enable
TPND	Timer 1 interrupt pending (timer underflow or capture edge)
C	Carry Flip/Flop

HC Half-Carry Flip/Flop

Table 12-3 PSW Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	TPND	ENTI	IPND	BUSY	ENI	GIE

The timer and MICROWIRE/PLUS control register bits are:

SL1 & SLO	Select the MICROWIRE/PLUS clock divide-by (00=2,01=4,1x=8)
IEDG	External interrupt edge polarity (0 = rising edge, 1 = falling edge)
MSEL	Selects G5 and G4 as MICROWIRE/PLUS signals SK and SO
TRUN	Used to start and stop the timer/counter (1 = run, 0 = stop)
TC1	Timer 1 Mode Control Bit
TC2	Timer 1 Mode Control Bit
TC3	Timer 1 Mode Control Bit

Table 12-4 CNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1	TC2	TC3	TRUN	MSEL	IEDG	SL1	SL0

12.10 MEMORY MAP

The COP8780 is based on a memory mapped architecture. All data memory, I/O ports, port registers and function registers are mapped into the data memory address space. Table 12-5 shows the organization of the data memory address space and the mapping of specific addresses. Read-only memory locations are noted in the table.

12.11 RESET

The following initializations are performed by the COP8780 at reset:

PORT C:	TRI-STATE
PORT D:	LOGIC HIGH
PORT G:	TRI-STATE
PORT L:	TRI-STATE
PC:	CLEARED
PSW and CNTRL:	CLEARED

Table 12-5 COP8780 Memory Map

ADDRESS	CONTENTS
00 to 2F 30 to 7F	48 on-chip RAM bytes* Unused RAM address space (reads as all 1's)*
00 to 6F 70 to 7F	112 on-chip RAM bytes** Unused RAM address space (reads as all 1's)**
C0 to CF	Reserved
D0 to DF	On-chip I/O and registers
D0	Port L data register
D1	Port L configuration register
D2	Port L input pins (read only)
D3	Reserved
D4	Port G data register
D5	Port G configuration register
D6	Port G input pins (read only)
D7	Port I input pins (read only)
D8	Port C data register
D9	Port C configuration register
DA	Port C input pins (read only)
DB	Reserved
DC	Port D
DD to DF	Reserved
E0 to EF	On-chip functions and registers
E0 to E8	Reserved
E9	MICROWIRE/PLUS shift register (SIOR)
EA	Timer lower byte
EB	Timer upper byte
EC	Timer autoload register lower byte
ED	Timer autoload register upper byte
EE	CNTRL control register
EF	PSW register
F0 to FF	16 on-chip RAM bytes mapped as registers
FC	X register
FD	SP register
FE	B register

* 64 on-chip RAM bytes selected in ECON register

** 128 on-chip RAM bytes selected in ECON register

B, X, SP:	UNKNOWN at power-on reset UNCHANGED at external reset
RAM:	UNKNOWN at power-on reset UNCHANGED at external reset
ACC and TIMER 1:	UNKNOWN at power-on reset UNKNOWN at external reset with Crystal oscillator clock option selected UNCHANGED at external reset with R/C or External oscillator clock options

12.12 OSCILLATOR CIRCUITS

Section 2.7 describes the three clock oscillator configurations available for the COP8780. The CKI 1 and CKI 2 bits in the ECON register are used to select the clock option. See Section 12.8 for more details.

12.13 PROGRAMMING THE COP8780C

Programming the COP8780 is accomplished through a National Semiconductor COP8 Duplicator Board. Third-party programming support is also available. Refer to the COP8780 datasheet for more information on third-party programming support.

The duplicator board is a stand-alone programmer capable of supporting all COP8780 package types when combined with available adaptor boards (Scrambler Boards). The duplicator works in conjunction with a pre-programmed source EPROM containing the application program. (The source EPROM may be programmed via a standard programmer.) The duplicator board essentially copies the information from the source EPROM into the COP8780 program memory.

In addition to the application program stored in locations 0000 through 0FFF Hex, the source EPROM must contain a value for the ECON register at location 1FFF Hex. The following tables provide examples of some ECON register values. For more detailed information, refer to Section 12.8.

Table 12-6 EPROM Security Enabled

RAM Memory	External CKI	RC Oscillator	Crystal Oscillator
64 Bytes	18	10	00
128 Bytes	1A	12	02

Table 12-7 EPROM Security Disabled

RAM Memory	External CKI	RC Oscillator	Crystal Oscillator
64 Bytes	38	30	20
128 Bytes	3A	32	22

12.14 ERASING THE COP8780C EPROM

The COP8780C EPROM program memory is erased by exposing the transparent window of the UV-erasable package to an ultraviolet light source. Erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000 Å to 4000 Å range.

After programming, opaque labels should be placed over the window of the device to prevent functional failure due to the generation of photo currents, erasure and excessive HALT current. Note that the device will also draw more current than normal (especially in HALT mode) when the window of the device is not covered with an opaque label.

The recommended erasure procedure for the COP8780 is exposure to short wave ultraviolet light with a wavelength of 2537 Å. The integrated dose (UV intensity X exposure time) for erasure should be at least 30 W-sec/cm^2.

The COP8780 device should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes, which should be removed before erasure. Refer to the COP8780 datasheet for information on the minimum erasure time for various light intensities.

An erasure system should be calibrated periodically. The distance from lamp to device should be maintained at one inch. The erasure time increases as a square of the distance. Lamps lose intensity as they age. When a lamp has aged, the system should be checked to make sure that adequate UV dosages are being applied for full erasure.

Common symptoms of insufficient erasure are:

- Inability to be programmed.
- Operational malfunctions associated with V_{CC}, temperature, or clock frequency.
- Loss of data in program memory.
- A change in configuration values in the ECON register.

12.15 EMULATION DEVICES

The COP8780C, COP8781C, and COP8782C can be used to emulate the COP820C, COP840C, and COP880C family microcontrollers. Table 12-8 shows which COP800 Basic Family members may be emulated by a COP8780.

Table 12-8 Emulation Cross Reference

Part Number	Package	Type	Emulates
COP8780CV	44 PLCC	OTP[a]	COP880C
COP8780CEL	44 LDCC	UV Erasable	COP880C
COP8780CN	40 DIP	OTP	COP880C
COP8780CJ	40 DIP	UV Erasable	COP880C
COP8781CN	28 DIP	OTP	COP881C COP840C COP820C
COP8781CJ	28 DIP	UV Erasable	COP881C COP840C COP820C
COP8781CWM	28 SO	OTP	COP881C COP840C COP820C
COP8781CMC	28 SO	UV Erasable	COP881C COP840C COP820C
COP8782CN	20 DIP	OTP	COP842C COP822C
COP8782CJ	20 DIP	UV Erasable	COP842C COP822C
COP8782CWM	20 SO	OTP	COP842C COP822C
COP8782CMC	20 SO	UV Erasable	COP842C COP822C

[a] One-Time Programmable

Chapter 13

APPLICATION HINTS

13.1 INTRODUCTION

This chapter describes several application examples using the COP800 family of microcontrollers. Design examples often include block diagrams and/or assembly code. Certain hardware design considerations are also presented.

Topics covered in this chapter include the following:

- MICROWIRE/PLUS implementation examples
- Timer application examples
- Triac control example
- COP820CJ design examples
- Programming examples (clear RAM, binary arithmetic)
- External power wakeup circuit
- External watchdog circuit
- Input protection on COP800 pins
- Electromagnetic interference (EMI) considerations

13.2 MICROWIRE/PLUS INTERFACE

A whole family of off-the-shelf devices is directly compatible with the MICROWIRE/PLUS interface. This allows direct interface of the COP800 microcontrollers with a large number of peripheral devices. The following sections provide examples of the MICROWIRE/PLUS interface. These examples include a master/slave mode protocol, code for a continuous mode of operation, code for a fast burst mode of operation, and a COP820 to an NMC93C06 interface.

13.2.1 MICROWIRE/PLUS Master/Slave Protocol

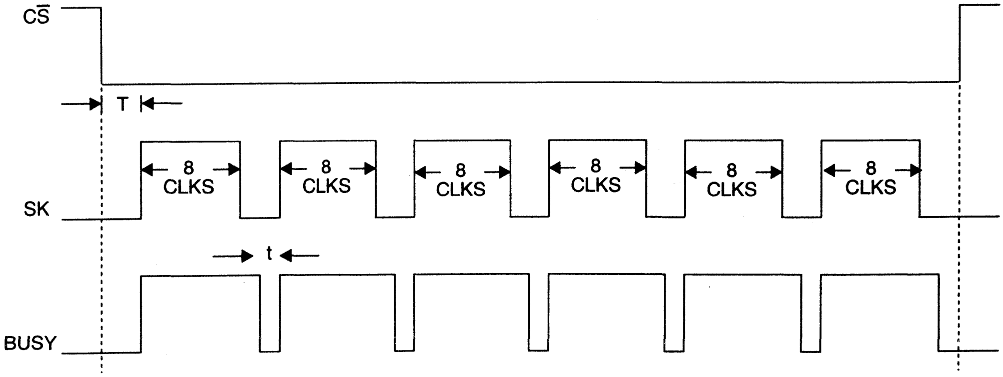
This section gives a sample MICROWIRE/PLUS master/slave protocol, the slave mode operating procedure for the sample protocol, and a timing illustration of the sample protocol.

1. CS from the master device is connected to G0 of the slave device. An active-low level on the CS line causes the slave to interrupt.

2. From the high-to-low transition on the CS line, there is no data transfer on the MICROWIRE interface until the setup time T has elapsed (see Figure 13-1).
3. The master initiates data transfer on the MICROWIRE interface by turning on the SK clock.
4. A series of data transfers takes place between the master and slave devices.
5. The master pulls the CS line high to end the MICROWIRE operation. The slave device returns to normal mode of operation.

Slave Mode Operating Procedure (for the previous protocol):

1. Set the MSEL bit in the CNTRL register to enable MICROWIRE; G0 and G5 are configured as inputs and G4 as an output.
2. Normal mode of operation until interrupted by CS going low.
3. Set the BUSY flag and load SIOR register with the data to be sent out on S0. (The shift register shifts eight bits of data from S0 at the high-order end of the shift register. Concurrently, eight new bits of data from SI are loaded into the low-order end of the shift register.)
4. Wait for the BUSY flag to be reset. (The BUSY flag automatically resets after 8 bits of data have been shifted.)
5. If data is being read in, the contents of the SIO register are saved.
6. The prearranged set of data transfers are performed.
7. Repeat steps 3 through 6. The user must ensure step 3 is performed within *t-time* (refer to Figure 13-1) as agreed upon in the protocol.



COP800-41

Figure 13-1 MICROWIRE/PLUS Sample Protocol Timing

13.2.2 MICROWIRE/PLUS Continuous Mode

The MICROWIRE/PLUS interface can be used in continuous clock mode with the master mode divide-by-eight clock division factor selected. The maximum data transfer rate for this MICROWIRE/PLUS continuous clock mode is 64 microseconds per byte (equivalent to 125 KHz) for parts operating with a 1 μsec instruction cycle.

The continuous clock mode is achieved by resetting the BUSY bit under program control just before it would automatically be reset with the hardware, and then immediately setting the BUSY bit with the next instruction. The SIO MICROWIRE shift register is then loaded (or read) with the following instruction. This loading of SIO occurs before the SK clock goes high, even though the previous set BUSY bit instruction has started the divide-by-eight (3-stage) counter. The B pointer must be already set up to point at the PSW register where the MICROWIRE BUSY bit is located. This three-instruction sequence is programmed as follows:

Instruction			Bytes/Cycles
RBIT	BUSY,	[B]	1/1
SBIT	BUSY,	[B]	1/1
X	A,	SIOR	2/3

This three-instruction sequence must be embedded in an instruction program loop that is exactly 64 instruction cycles (t_c cycles) in length. This yields a 125-KHz (64 microseconds per byte) data transfer rate at the maximum instruction cycle rate of 1 MHz.

The following program demonstrates the use of the MICROWIRE/PLUS continuous clock mode. The program continually outputs the 256 bytes of the current program memory block on the MICROWIRE S0 output pin (G4). The low-order bit (G0) of Port G is set to cover the transition period of the three-instruction sequence outlined previously, where the SIO register is loaded with a new byte.

```
PORTGD   = 0D4
PORTGC   = 0D5
SIOR     = 0E9
CNTRL    = 0EE
PSW      = 0EF
MWTEMP   = 0F0
MWCNT    = 0F1

MARK     = 0
BUSY     = 2
                                                        CYCLES
MWCONT:  LD       PORTGC, #031
         LD       PORTGD, #0
         LD       CNTRL, #0B
         LD       B, #PSW
         LD       MWCNT, #0
MWLOOP:  LD       A, MWCNT
         INC      A                                        3
         X        A, MWCNT                                 1
         LAID                                              3
         SBIT     MARK, PORTGD                             3
         RBIT     BUSY, [B]                                4
         SBIT     BUSY, [B]                                1
         X        A, SIOR                                  1
         RBIT     MARK, PORTGD                             3
         LD       MWTEMP, #6                               4
MWLUP:   DRSW     MWTEMP                    3 ⎫            3
         JP       MWLUP                       ⎬ x6-2=     34
         NOP                                3 ⎭            1
         JP       MWLOOP                                   3
                                                      ---------
         TOTAL CYCLES IN MWLOOP =                         64
```

13.2.3 MICROWIRE/PLUS Fast Burst Output

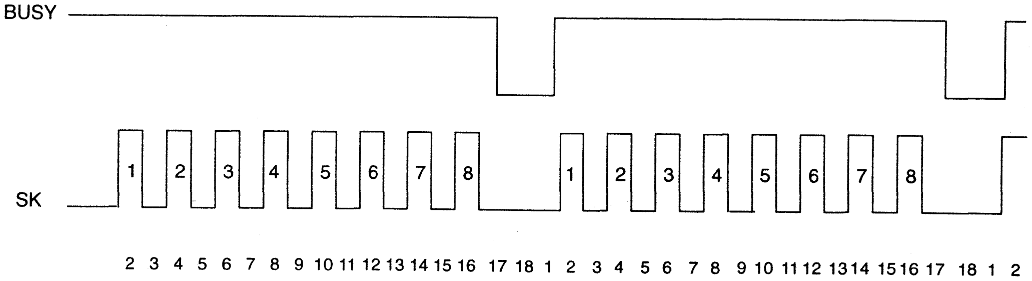
The maximum COP800 MICROWIRE/PLUS master mode burst clock rate (using the divide-by-two clock division factor) is 500 KHz. This assumes that the COP800 microcontroller is running at the maximum instruction cycle frequency of 1 MHz. The equivalent time of one extra master mode SK clock cycle is necessary to set up the next byte (and/or read the previous byte) in SIOR when using the burst mode SK frequency. This yields an equivalent minimum data transfer time of 18 microseconds per byte.

The following program demonstrates the use of the MICROWIRE/PLUS burst clock mode at the maximum data transfer rate (with the divide-by-two master mode clock option selected). The X pointer is initialized to the TOP of a RAM table, where SIZE represents the size of the table. This subroutine outputs the contents of the RAM table on the MICROWIRE S0 output pin (G4).

Register Definitions

```
SIOR     = 0E9
CNTRL    = 0EE
PSW      = 0EF
MWCNT    = 0F0

BUSY     =2
                                              INSTRUCTION  CYCLE NUMBER
                                                 CYCLES    IN Figure 13-2
MWBRST:  LD      CNTRL, #8
         LD      B, #PSW
         LD      MWCNT, #SIZE
         LD      X, #TOP
MWLOOP:  LD      A, [X-]                           3          13, 14, 15
         X       A, SIOR                           3          16, 17, 18
         SBIT    BUSY, [B]                         1              1
         NOP*                                      1              2
         NOP*                                      1              3
         LAID*                                     3           4, 5, 6
         DRSZ    MWCNT                             3           7, 8, 9
         JP      MWLOOP                            3          10, 11, 12
         RET

         TOTAL CYCLES IN MWLOOP =                 18
* Time Delay
```

The MICROWIRE BUSY bit is allowed to reset automatically with the hardware following the eighth SK clock. The transfer of new data into the SIO register and the transfer of the new input data from SIO to A occurs at the end of the second cycle of the three-cycle "exchange A with SIO" instruction. This exchange instruction is immediately followed by the "set BUSY bit" instruction to initiate another MICROWIRE serial byte transfer. The associated timing for this 18-instruction cycle MICROWIRE loop is shown in Figure 13-2.



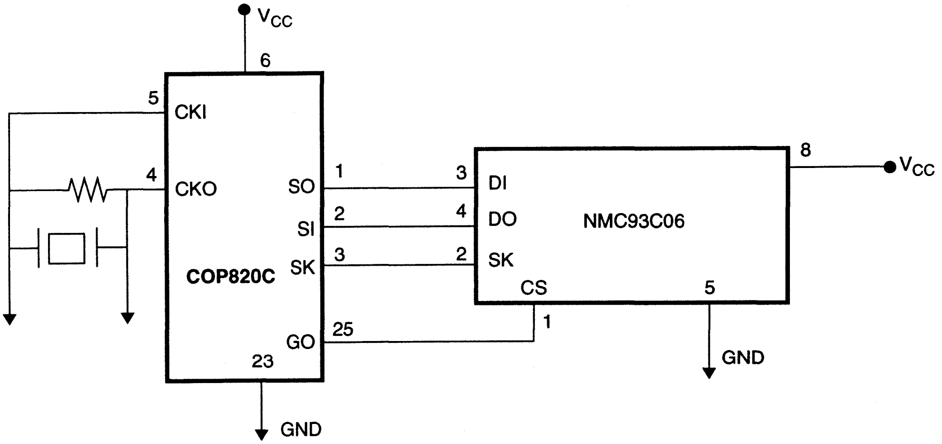
COP800-42

Figure 13-2 MICROWIRE/PLUS Fast Burt Timing

13.2.4 NMC93C06-COP820C Interface

This example shows the COP820 interface to a NMC93C06, a 256-bit E^2PROM, using the MICROWIRE interface. The pin connection involved in interfacing a NMC93C06 with the COP820C microcontroller is shown in Figure 13-3. Some notes on the NMC93C06 interface requirements are:

1. The SK clock frequency should be less than 250 KHz.
2. CS low period following an Erase/Write instruction must not exceed 30 ms maximum. It should be set at typical or minimum specification of 10 ms.
3. The start bit on DI must be set by a "0" to "1" transition following a CS enable ("0" to "1") when executing any instruction. One CS enable transition can execute only one instruction.
4. In the read mode, following an instruction and data train, the DI is a "don't care" while the data is being output for the next 17 bits or clocks. The same is true for other instructions after the instruction and data has been fed in.
5. The data out train starts with a dummy bit 0 and is terminated by chip deselect. Any extra SK cycle after 16 bits is not essential. If CS is held on after all 16 of the data bits have been output, the DO will output the state of DI until another CS low to high transition starts a new instruction cycle.
6. After a read cycle, the CS must be brought low for one SK clock cycle before another instruction cycle starts.



COP800-32

Figure 13-3 NMC93C06-COP820C Interface

The following table describes the instruction set of the NMC93C06. In the table A3A2A1A0 corresponds to one of the sixteen 16-bit registers.

Commands	Start Bit	Opcode	Address	Comments
READ	1	0000	A3A2A1A0	Read Register 0–15
WRITE	1	1000	A3A2A1A0	Write Register 0–15
ERASE	1	0100	A3A2A1A0	Erase Register 0–15
EWEN	1	1100	0001	Write/Erase Enable
EWDS	1	1100	0010	Write/Erase Disable
WRAL	1	1100	0100	Write All Registers
ERAL	1	1100	0101	Erase All Registers

All commands, data in, and data out are shifted in/out on the rising edge of the SK clock. All instructions are initiated by a low-to-high transition on CS followed by a low-to-high transition on DI.

A detailed explanation of the NMC93C06 E^{2}PROM timing, instruction set, and other considerations can be found in the datasheet. A source listing of the software to interface the NMC93C06 with the COP820C is provided below.

```
.INCLD COP820.INC
;
;This program provides in the form of subroutines, the ability to erase,
;enable, disable, read and write to the NMC93C06 EEPROM.
;
;
SNDBUF = 0                     ;Contains the command byte to be written to NMC93C06
RDATL = 1                      ;Lower byte of the NMC93C06 register data read
RDATH = 2                      ;Upper byte of the NMC93C06 register data read
WDATL = 3                      ;Lower byte of the data to be written to NMC93C06
                               ;register
ADRESS = 5                     ;The lower 4-bits of this location contain the
                               ;address of the NMC93C06 register to read/write
FLAGS = 6                      ;Used for setting up flags
                               ;
                               ;Flag value        Action
                               ;00                Erase, enable, disable, erase all
                               ;01                Read contents of NMC93C06 register
                               ;03                Write to NMC93C06 register
                               ;Others            Illegal combination
DLYH = 0F0
DLYL = 0F1
;
;The interface between the COP820C/840C and the NMC93C06 (256-bit EEPROM) consists of
;four lines: The G0 (chip select line), G4 (serial out SO), G5 (serial clock SK), and
;G6 (serial in SI).
```

```
;
; Initialization
;
        LD      PORTGC,#031 ;Setup G0, G4, G5 as outputs
        LD      PORTGD,#00  ;Initialize G data reg to zero
        LD      CNTROL,#08  ;Enable MSEL, select MW rate of 2tc
        LD      B,#PSW
        LD      X,#SIOR

;
;This routine erases the memory location pointed to by the address contained in the
;location "ADRESS." The lower nibble of "ADRESS" contains the NMC93C06 register
;address and the upper nibble should be set to zero.
;
ERASE:   LD      A,ADRESS
         OR      A,#0C0
         X       A,SNDBUF
         LD      FLAGS,#0
         JSR     INIT
         RET
;
;This routine enables programming of the NMC93C06. Programming must be preceded once
;by a programming enable (EWEN).
;
EWEN:    LD      SNDBUF,#030
         LD      FLAGS,#0
         JSR     INIT
         RET
;
This routine disables programming of the NMC93C06
;
EWDS:    LD      SNDBUF,#0
         LD      FLAGS,#0
         JSR     INIT
         RET
;
;This routine erases all registers of the NMC93C06
;
ERAL:    LD      SNDBUF,#020
         LD      FLAGS,#0
         JSR     INIT
         RET
;
;This routine reads the contents of the NMC93C06 register. The NMC93C06 address is
;specified in the lower nibble of location "ADRESS." The upper nibble should be set
;to zero. The 16-bit contents of the NMC93C06 register are stored in RDATL and RDATH.
;
READ:    LD      A,ADRESS
         OR      A,#080
         X       A,SNDBUF
         LD      FLAGS,#1
         JSR     INIT
         RET
;
;This routine writes a 16-bit value stored in WDATL and WDATH to the NMC93C06 register
;whose address is contained in the lower nibble of the location "ADRESS." The upper
;nibble of address location should be set to zero.
;
WRITE:   LD      A,ADRESS
         OR      A,#040
         X       A,SNDBUF
```

```
          LD        FLAGS,#3
          JSR       INIT
          RET
;
;This routine sends out the start bit and the command byte. It also deciphers the
;contents of the flag location and takes a decision regarding write, read or return
;to the calling routine.
;
INIT:     SBIT      0,PORTGD             ;Set chip select high
          LD        SIOR,#001            ;Load SIOR with start bit
          SBIT      BUSY,[B]             ;Send out the start bit
PUNT1:    IFBIT     BUSY,[B]
          JP        PUNT1
          LD        A,SNDBUF
          X         A,[X]                ;Load SIOR with command byte
          SBIT      BUSY,[B]             ;Send out command byte
PUNT2:    IFBIT     BUSY,[B]
          JP        PUNT2
          IFBIT     0,FLAGS              ;Any further processing?
          JP        NOTDON               ;Yes
          RBIT      0,PORTGD             ;No, reset CS and return
          RET
;
NOTDON:   IFBIT     1,FLAGS              ;Read or write?
          JP        WR494                ;Jump to write routine
          LD        SIOR,#000            ;No, read NMC93C06
          SBIT      BUSY,PSW             ;Dummy clock to read zero
          RBIT      BUSY,[B]
          SBIT      BUSY,[B]
PUNT3:    IFBIT     BUSY,[B]
          JP        PUNT3
          X         A,[X]
          SBIT      BUSY,[B]
          X         A,RDATH
PUNT4:    IFBIT     BUSY,[B]
          JP        PUNT4
          LD        A,[X]
          X         A,RDATL
          RBIT      0,PORTGD
          RET
;
WR494:    LD        A,WDATH
          X         A,[X]
          SBIT      BUSY,[B]
PUNT5:    IFBIT     BUSY,[B]
          JP        PUNT5
          LD        A,WDATL
          X         A,[X]
          SBIT      BUSY,[B]
PUNT6:    IFBIT     BUSY,[B]
          JP        PUNT6
          RBIT      0,PORTGD
          JSR       TOUT
          RET
;
;Routine to generate delay for write
;
TOUT:     LD        DLYH,#00A
```

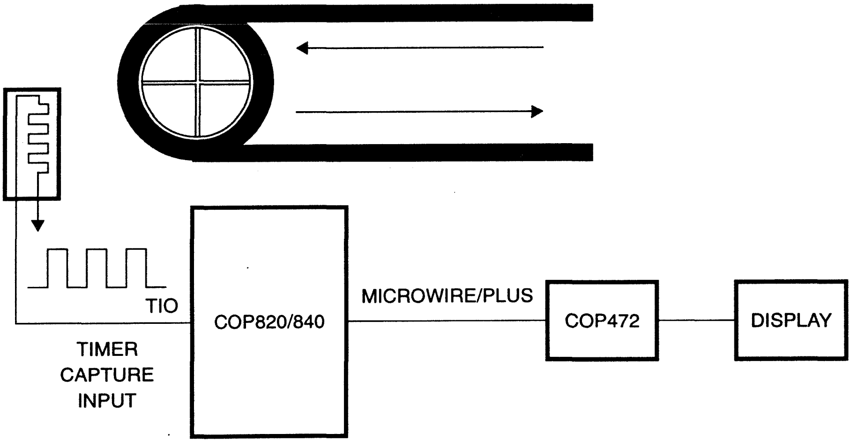
```
WAIT:     LD        DLYL,#0FF
WAIT1:    DRSZ      DLYL
          JP        WAIT1
          DRSZ      DLYH
          JP        WAIT
          RET
          .END
```

13.3 TIMER APPLICATIONS

This section describes some applications using the on-chip timer: speed measurement using the Input Capture mode, a simple D/A converter using the PWM mode, and an external event counter using the External Event Counter mode.

13.3.1 Timer Capture Example

The Timer Input Capture Mode can be used to measure the time between events. The simple block diagram in Figure 13-4 shows how the COP820/840 can be used to measure motor speed based on the time required for one revolution of the wheel. A magnetic sensor is used to produce a pulse for each revolution of the wheel.



COP800-33

Figure 13-4 Timer Capture Application

In the capture mode of operation, the timer counts down at the instruction cycle rate. In this application, the timer is set up to generate an interrupt on a TIO positive edge transition. The timer is initialized to 0FFFF Hex and begins counting down. An edge transition on the TIO input pin of the timer causes the current timer value to be copied into the RA register. In addition, it sets the timer interrupt pending flag, which causes a program branch to memory location 0FF Hex. The interrupt service routine for the timer is stored at that program memory location. The interrupt service routine resets the timer interrupt pending flag. It then reads the contents of RA and stores it in RAM for later processing. An RETI instruction is used to return to normal program execution and re-enable subsequent interrupts (by setting the GIE bit).

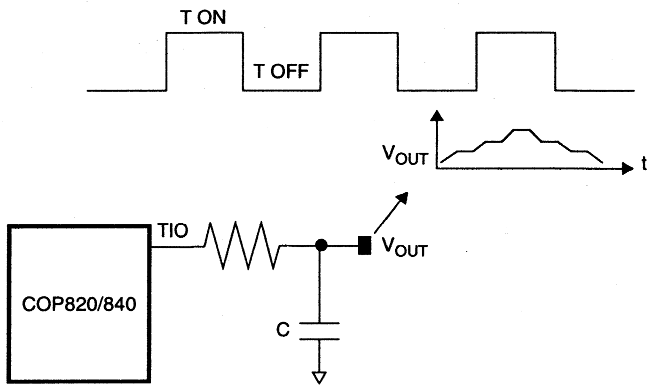
On the next rising edge transition on TIO, the program returns to the interrupt service routine. The value in RA is read again, and compared with the previously read value. The difference between the two captured values, multiplied by the instruction cycle time, gives the time for one revolution. This can be easily converted to a frequency. The frequency may be displayed on an LCD using the COP MICROWIRE/PLUS interface and a COP472-3 LCD Driver.

An example of the code that can be used for this application is provided below.

```
         PSW      = 0EF
         CNTRL    = 0EE
         TPND     = 5
         TRUN     = 4
         PORTGC   = 0D5
         PORTGD   = 0D4

         LD       PORTGC,#00        ;Configure G3/TIO as input
         LD       PORTGD,#08        ;Weak pull-up on G3
         LD       CNTRL,#0C0        ;Timer as capture mode, positive edge
         LD       PSW,#011          ;Enable timer and global interrupts
         LD       TMRLO,#0FF        ;Timer lower byte
         LD       TMRHI,#0FF        ;Timer upper byte
         LD       TRALO,#0FF        ;Auto-reload lower byte
         LD       TRAHI,#0FF        ;Auto-reload upper byte
         SBIT     TRUN,CNTRL        ;Start timer
SELF:    JP       SELF              ;Wait for capture

;Timer interrupt handling routine

.=0FF
         IFBIT    TPND,PSW          ;Is it timer interrupt?
         JP       TIMSERV           ;Yes
         JP       Error             ;No go to error routine

;Timer service routine

TIMSERV: RBIT     TPND,PSW          ;Reset pending flag
         .                          ;(Process Timer Capture)
         .
         .
         RETI                       ;Return from interrupt
.END
```

13.3.2 Timer PWM Example

Figure 13-5 shows how a minimal-component D/A converter can be built out of the timer register pair in the auto-reload mode. The timer is placed in PWM mode and initialized with the ON time, and register RA (the auto-reload register) is initialized to the OFF time. TIO/G3 is configured as an output and preset to logic high. By setting the TRUN bit in the CNTRL register, the timer starts and counts down at the instruction cycle rate when an underflow of the timer occurs, the TIO output pin is toggled, the contents of the RA register (OFF time) are copied into the timer, and the TPND bit in PSW register is set. A timer underflow also generates a timer interrupt, where program control vectors to program memory address 0FF Hex. The interrupt service routine at that address resets the timer interrupt pending flag and alternately replaces the value in the RA register with either the OFF time or the ON time after each timer underflow. A PWM signal appears on the TIO output pin.



A simple D-A converter using the timer to generate a PWM output.

COP800-34

Figure 13-5 PWM Timer Application

With the ON time set equal to the OFF timer (50% duty cycle), the capacitor charges and discharges slightly in each cycle, and the output remains at a fairly constant level. With the duty cycle set larger or smaller than 50%, the capacitor gains or loses charge, and the output voltage rises or falls.

An example of the code that can be used for this application is provided below.

```
;Operating the timer in PWM mode
;CONSTANT DECLARE
FLAG     = 05
ONFLG    = 0
ONTMHI   = 07
ONTMLO   = 0FF
OFTMHI   = 0F
OFTMLO   = 0FF

;Timer auto-reload mode running off internal clock.
;Interrupts are used.
;The output is a duty cycle output on G3.
;Timer logic automatically toggles G3.
;
PWMI:
         LD       FLAG,#00          ;Clear ONFLG i.e., start with off time
         LD       CNTRL1,#0A0       ;Timer stopped, G3 enabled, AR mode
         LD       PORTGD,#00        ;SBIT G3 low
         LD       PORTGC,#08        ;SBIT G3 as an output/TIO
         LD       TMRLO,#OFTMLO     ;Initialize timer lower byte
```

```
          LD        TMRHI,#OFTMLO       ;Timer upper byte
          LD        TAUHI,#ONTMHI       ;Auto-reload register upper byte
          LD        TAULO,#ONTMLO       ;Auto-reg. lower byte with on time
          SBIT      TRUN,CNTRL          ;Start timer
          LD        PSW,#011            ;Enable timer interrupts
;
SLEEP:
          JP        SLEEP               ;Wait for timer underflow
;
;The interrupt routine below handles timer interrupts
          .= X'00FF
TINTR:
          RBIT      TPND,PSW            ;Got it, RBIT for future
          IFBIT     FLAG,ONFLG          ;On time?
          JP        SONT                ;Need to set bit up for on time
SOFT:
          LD        T1RALO,#OFTMLO      ;Off time
          LD        T1RAHI,#OFTMHI
          SBIT      FLAG,ONFLG
          RETI
;
SONT:
          LD        T1RALO,#ONTMLO      ;On time
          LD        T1RAHI,#ONTMHI
          RBIT      FLAG,ONFLG
          RETI
          .END
```

13.3.3 External Event Counter Example

This mode of operation is very similar to the PWM Mode of operation. The only difference is that the timer is clocked from an external source. This mode provides the ability to perform control of a system based on counting a predetermined number of external events, such as searching for the nth sector on a disk or testing every nth part on an assembly line. The code for this example is provided below.

```
; Operating the timer in External Event Counter Mode
          PSW       = 0EF
          CNTRL     = 0EE
          TPND      = 5
          TRUN      = 4
          PORTGC    = 0D5
          PORTGD    = 0D4

          RBIT      3,PORTGC            ;Configure G3/TIO as Hi-Z input
          RBIT      3,PORTGD            ;
          LD        CNTRL,#00           ;Select timer as external event counter
          LD        PSW,#011            ;Enable timer and global interrupts
          LD        TMRLO,#COUNT0       ;Timer lower byte
          LD        TMRHI,#COUNT1       ;Timer upper byte
          LD        TRALO,#Count0       ;Auto-reload lower byte
          LD        TRAHI,#Count1       ;Auto-reload upper byte
          SBIT      TRUN,CNTRL          ;Start timer
SELF:     JP        SELF                ;Wait for the n-th count
```

```
;Timer interrupt handling routine

.=0FF
         IFBIT    TPND,PSW           ;Is it timer interrupt?
         JP       TIMSERV            ;Yes
         JP       ERROR              ;No go to error routine

;Timer service routine

TIMSERV: RBIT     TPND,PSW           ;Reset pending flag
         RBIT     TRUN,PSW           ;Stop timer
         .                           {Process timer}
         .
         .
         .
         .
         SBIT     TRUN,PSW           ;Start timer
         RETI                        ;Return from interrupt
.END
```

13.4 TRIAC CONTROL

The COP800 family devices provide computational ability and speed which is suitable for intelligently managing power control. In order to intelligently control a triac on a cyclic basis, an accurate time base must be established. This may be in the form of an AC 60Hz sync pulse generated by a zero voltage detection circuit or a simple real-time clock. The COP800 family is suited to accommodate either of these time base schemes while accomplishing other tasks.

Zero voltage detection is the most useful scheme in AC power control because it affords a real-time clock base as well as a reference point in the AC waveform. With this information it is possible to minimize RFI by initiating power-on operations near the AC line voltage zero crossing. It is also possible to fire the triac only a portion of the cycle, thus utilizing conduction angle manipulation. This is useful in both motor control and light-intensity control.

COPS software is capable of compensating for noisy or semi-accurate zero voltage detection circuits. This can be accomplished by using delays and debounce algorithms in the software. With a given reference point in the AC waveform, it becomes easy to divide the waveform to efficiently allocate processing time.

These techniques are demonstrated in the code listing below. This application example is based on the half cycle approach of AC power for triac light intensity control. The code will intensify and deintensify the lamp under program control.

This program example is not intended to be a final functional program. It is a general-purpose light intensifying/deintensifying routine which can be modified for a light dimmer application. The delay routines are based on a 10 MHz crystal clock (1 s instruction cycle). The COP820C's 16-bit timer can be used for timing the half cycle of an AC power line, and the timer can be started or stopped under software control. Timer T1 is a read/write memory mapped counter with an associated 16-bit auto-reload register. Zero crossings of the 60 Hz line are detected and software debounced to initiate each half cycle, so the triac is serviced on every half cycle of the power line. This program divides the half cycle of a 60 Hz AC power line into 16 levels. Intensity is varied by increasing or decreasing the conduction angle by firing the triac at various levels. Each level is a fixed time which is loaded into the timer. Once a true zero cross is detected, the timer starts and triac is serviced.

A level/sublevel approach is utilized to vary the conduction angle and to provide a prolonged intensifying period. The maximum intensity reached is at the maximum conduction angle (level), and the time required to get to that level is loaded into the timer in increments. Once a level has been specified, the remaining time in the half cycle is then divided into sublevels. The sublevels are increased in steps to the maximum level and the triac is fired 16 times per sublevel, thus creating the intensity time base. For deintensifying, the sublevels are decremented.

```
NATIONAL SEMICONDUCTOR CORPORATION
COP800 CROSS ASSEMBLER, REV:E,22 JUN 90
TIMER

            ;_______________________________________________________
            ;
            ;  THIS IS A GENERAL PURPOSE LIGHT DIMMER PROGRAM
            ;  USING COP800 TIMER WRITTEN BY FARID NOORY JULY 1990
            ;  IT USES A 10 MHZ CLOCK (1 us INSTRUCTION CYCLE TIME)
            ;
            ;_______________________________________________________

                     .INCLD COP820.INC
                     .TITLE TIMER, 'TIMER APPLICATION EXAMPLE'
                     ;INITIALIZATIONS
     00F0            TEMP  = 0F0
     00F1            LEVEL = 0F1
     00F2            FIN   = 0F2         ;FIRE #
     00F3            REG1  = 0F3
0002 D200            LD      FIN,#000    ;
0004 D140            LD      LEVEL,#040  ;SUBLEVEL
0006 BCD500          LD      PORTGC,#000 ;MAKE G PORT AS INPUT
0009 BCD404          LD      PORTGD,#004 ;WITH WEAK PULL-UP
000C BCEE80          LD      CNTRL,#080  ;TIMER AS AUTORELOAD
000F BCEF00          LD      PSW,#000    ;
0012 BCEA7D          LD      TMRLO,#07D  ;TIMER AND AUTORELOAD REG
0015 BCEB00          LD      TMRHI,#000  ;INITIALIZED TO .5ms DELAY
0018 BCECE8          LD      TAULO,#0EB  ;EACH
0018 BCED03          LD      TAUHI,#003

            ; POWER UP SYNCHRONIZATION OR RESET SYNCH.
            ;
001E BDD672 BEG:     IFBIT   2,PORTGP    ;IF BIT G2 =1
0021 01              JP      HI
0022 FB              JP      BEG         ;TO SYNC. UP 60HZ
0023 BDD672 HI:      IFBIT   2,PORTGP    ;IS IT STILL ONE
0026 FC              JP      HI          ;YES WAIT TILL ITS ZERO


            ;_______________________________________________________
            ;TEST FOR TRUE ZERO CROSS (Valid Transition)
            ;_______________________________________________________
            ;HERE WE PROVIDE DEBOUNCE FOR ZERO CROSS DETECTION
            ;_______________________________________________________
                                         ;START OF DEBOUNCE DELAY
                                         ;IF BIT G2 = 0
0027 3077            JSR     DELAY       ;TEMPORARY DELAY
0029 BDD672          IFBIT   2,PORTGP    ;WAS IT FALLS?
002C F1              JP      BEG         ;YES :FALLS ALARM
002D 203B   R DOIT:  JMP     INIT        ;NO : START!
002F BDD672 LO:      IFBIT   2,PORTGP    ;DEBOUNCE 0 TO 1
0032 01              JP      D1          ;IF 1 GO TO DELAY
0033 FB              JP      LO          ;IF NOT WAIT FOR A 1
0034 3077   D1:      JSR     DELAY       ;WE HAVE A CLEAN TRANSITION
0036 BDD672          IFBIT   2,PORTGP    ;IS IT STILL 1?
0039 F3              JP      DOIT        ;
003A F4              JP      LO          ;YES GO TO MAIN ROUTINE
            ;                            ;NO KEEP DELAY GOING NOISE
            ;*******************************************************
            ;
            ;         MAIN ROUTINE FOR INTENSIFY/DE-INTENSIFY
            ; THE PROGRAM GETS HERE WHEN A TRUE ZERO IS DETECTED
            ;*******************************************************   CYCLE TIME
003B 3098   INIT:    JSR     TIMER       ;DELAY FOR 1ms TO GET TO MAX  ;2/5
003D 9DF2            LD      A,FIN       ;CONTAINS FIR NUMBER          ;2/3
003F 9215            IFEQ    A,#015      ;ARE WE AT 15?                ;2/2
0041 04              JP      THER                                      ;1/3
0042 8A     BEGG:    INC     A           ;NO                           ;1/1
0043 9CF2            X       A,FIN       ;INC. THE FIRE #              ;2/3
```

```
0045 19                JP      FIRE           ;KEEP FIRING                     ;1/3
0046 D200      THER:   LD      FIN,#000                                        ;3/3
0048 9DF1              LD      A,LEVEL        ;YES NEXT LEVEL                  ;2/2
004A 8B                DEC     A                                               ;1/1
004B 9CF1              X       A,LEVEL        ;RESTORE LEVEL                   ;2/3
004D 9DF1              LD      A,LEVEL        ;GET BACK A
004F 9200              IFEQ    A,#000         ;IF MAX LEEL #HAS REACHED        ;2/2
0051 01                JP      LP2
0052 03                JP      LP3
0053 D140      LP2:    LD      LEVEL, #040    ;SET LEVEL                       ;2/2
0055 09                JP      FIRE           ;EXIT
0056 BDF175    LP3:    IFBIT   5,LEVEL        ;TEST WHICH LEVEL?
0059 308A              JSR     ADD            ;IF MAX NOT YET REACHED ADD DELAY
005B 307C              JSR     SUB            ;IF IT HAS SUBSTRACT DELAY
005D B8                NOP
005E B8                NOP
               ;*****************************************************
               ;                       SUBROUTINES                  *
               ;*****************************************************
005F BCDCFF    FIRE:   LD      PORTD,#0FF     ;PULL UP D PORT FOR              ;3/3
0062 9CF0              X       A,TEMP         ;32U SEC                         ;2/3
0064 64                CLR     A                                               ;1/1
0065 8A        LP6:    INC     A                                               ;1/1
0066 9203              IFEQ    A,#03                                           ;1/2
0068 01                JP      LP5                                             ;1/3
0069 FB                JP      LP6                                             ;1/3
006A 64        LP5:    CLR     A                                               ;1/1
006B BCDC00            LD      PORTD,#00      ;PULL D PORT LO                  ;3/3
006E 9CF0      TWO:    X       A,TEMP         ;RESTORE A                       ;2/3
0070 BDD672    HI1:    IFBIT   2,PORTGP       ;TEST FOR WHICH DEBOUNCE         ;1/1
0073 2023              JMP     HI             ;NEEDED                          ;2/3
0075 202F              JMP     LO                                              ;2/3


0077 D30F      DELAY:  LD      REG1,#00F                                       ;1/1
0079 C3        LOOP:   DRSZ    REG1                                            ;1/1
007A FE                JP      LOOP                                            ;1/3
007B 8E                RET                    ;FOR DEBOUNCING                  ;1/5
               ;DECREMENT THE TIMER BY THE DESIRED DELAY
               ;
007C 9DEC      SUB:    LD      A, TAULO
007E 917D              SUBC    A,#07D
0080 9CEC              X       A,TAULO
0082 9DED              LD      A,TAUHI
0084 9100              SUBC    A,#000
0086 A0                RC
0087 9CED              X       A,TAUHI
0089 8E                RET
               ;INCREMENT THE TIMER VALUE BY THE DESIRED DELAY
               ;
008A 9DEC      ADD:    LD      A, TAULO
008C 907D              ADC     A,#07D
008E 9CEC              X       A,TAULO
0090 9DED              LD      A,TAUHI
0092 9000              ADC     A,#000
0094 A0                RC
0095 9CED              X       A,TAUHI
0097 8D                RETSK

               TIMER:
0098 BDEE7C            SBIT    TRUN,CNTRL     ;START THE TIMER
009B BDEF75    LP1:    IFBIT   TPND,PSW       ;CHECK FOR TIMER UNDERFLOW
009E 01                JP      LP4
009F FB                JP      LP1
00A0 BDEE6C    LP4:    RBIT    TRUN,CNTRL     ;STOP THE TIMER
00A3 BDEF6D            RBIT    TPND,PSW       ;RESET THE UNDERFLOW FLAG
00A6 8E                RET
                       .END
```

13.5 COP820CJ APPLICATION HINTS

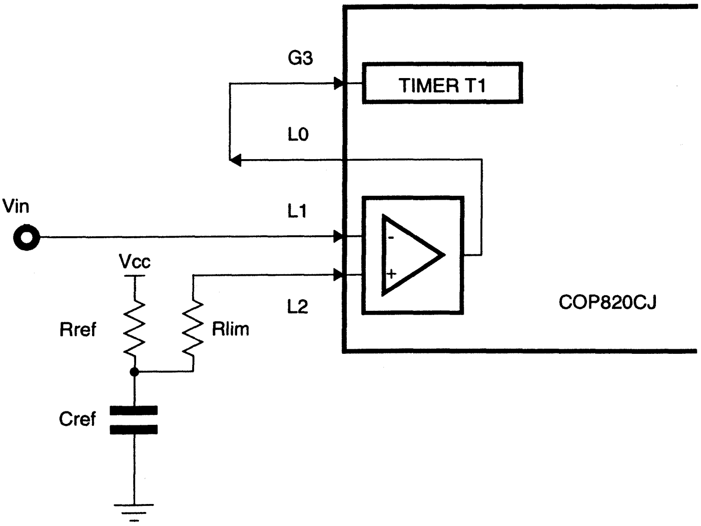
This section gives suggestions on how and where the COP820CJ special features may be used. Examples of how to use these features to implement system functions are given, followed by an example of an application which uses the feature.

13.5.1 Analog To Digital Conversion Using On-chip Comparator

Some microcontroller applications require a low-cost, but effective way of performing analog to digital conversion. A number of techniques for doing this are described in COP NOTE 1: “Analog to Digital Conversion Techniques with COPS Family Microcontrollers” and in Application Note 607: “Pulse Width Modulation A/D Conversion Techniques with COP800 Family Microcontrollers”. This section explains how the COP820CJ comparator can be integrated into two of the solutions described in these notes: the single slope A/D conversion technique and the pulse width modulation A/D technique.

Figure 13-6 shows the hardware connections for either type of A/D conversion technique. The voltage to be measured, V_{IN}, is connected to the inverting terminal of the comparator, pin L1. The non-inverting terminal, pin L2, is connected to an RC network via a current-limiting resistor. For the single slope technique, the comparator output on pin L0 is connected to the Timer T1 input pin G3. This is not required for the pulse width modulation technique.

The principle of the single slope conversion technique is to measure the time it takes for the RC network to charge up to the voltage level on the inverting terminal, by using Timer T1 in the input capture mode. The cycle count obtained in Timer T1 can be converted into real time if it is scaled by the COP8 clock frequency. If the COP8 is clocked



COP800-24

Figure 13-6 A/D Conversion Using COP820CJ Comparator and Timer T1

by a crystal, this parameter is known very accurately. Applications connected to the power line using an RC clock can use the line frequency as a reference with which to measure the RC clock. The time measurement is then converted into the voltage, either by direct calculation or by using a suitable approximation.

This very low cost technique is ideally suited to cost-sensitive applications which do not require high accuracy. The pulse width modulation A/D conversion technique will improve the accuracy at the cost of a higher conversion time. Application Note 607 describes this technique in detail.

The accuracy can be improved further by using a low-cost MM74HC4016 to multiplex the analog input voltage with an accurate voltage reference used for calibration. Replacing the resistor in the RC network with a current source will linearize the charging curve, offering better resolution.

The user must ensure that the input voltage supplied to the comparator lies within its input common mode range, which is shown in the characterization curves in the datasheet. This data shows that the input common mode range goes down to 0V if V_{CC} exceeds 4V and the magnitude of the offset voltage specification is relaxed to 25mV. The user must ensure that V_{IN} does not exceed the maximum input common mode range voltage during measurement.

Before the start of conversion, the capacitor must be discharged. The program must reconfigure pin L2 as an output logic low to perform the discharge. Timer T1 must be stopped and configured into input capture mode on a low-to-high transition. The T1 timer register must be cleared and pin G3 set up as a Hi-Z input. The comparator initialization described in Chapter 11 must also be performed. The conversion is started by starting timer T1 and then converting pin L2 back to an input.

The initial value of the comparator is zero. A capture event will occur when the RC voltage rises above the input voltage. If desired, the Timer T1 interrupt can be enabled to produce an interrupt on this capture event. The capture time can then be read and converted into voltage. This measurement technique has a resolution of 8 bits if the value of the timer is scaled to contain 1000 (or more) counts after five RC periods. The accuracy is primarily dependent on the accuracy of the user's estimation of the RC time constant, the offset voltage, and the user's approximation routine.

The following code example demonstrates how this is achieved in assembly code. In this example, polling the Timer T1 pending flag is used instead of interrupts. The 16-bit timer value is stored in REFHI:REFLO.

```
START:

; Discharge the capacitor by setting pin L2 to logic low and waiting.

           LD PORTLC,#004          ; Pin L2 is set to an output, logic low
           LD PORTLD,#000          ; to discharge the capacitor

DELAY:
           LD R0,#020              ; Delay depending on current limiting
DR0:       DRSZ R0                 ; resistor
           JP DR0

; Set up the comparator.

           LD PORTLC,#001          ; Pin L0 is an output, pins L1 and L2 are
                                   ; inputs
           LD PORTLD,#000
           SBIT CMPEN,CNTRL2
           SBIT CMPOE,CNTRL2

; Set up Port G as a Hi-Z input port

           LD PORTGD,#000          ; Port G is an 8 bit input port
           LD PORTGC,#000          ; with the weak pull-ups disabled

; Pre-load the timer with FFFF hex

           LD B,#TMRLO             ; Save bytes by using register B
           LD [B+],#0FF            ; Pre-load the timer with FFFF hex
           LD [B],#0FF

; Charge up the capacitor through pin L5 and start the timer

           LD B,#CNTRL1            ; Save bytes by using register B
           LD [B+],#0D0            ; Rising edge input capture, start Timer 1
                                   ; B now points to PSW
           RBIT TPND,[B]           ; Ensure that the pending flag is zero

; Wait until the first capture and save the captured value in REFHI:REFLO

WAITC1:    IFBIT TPND,[B]          ; Wait until the first capture
           JP STORE1
           JP WAITC1

STORE1:    RBIT TPND,[B]           ; Reset the pending flag
           LD X,#TAULO             ; Save bytes by using register X
           LD A,[X+]               ;
           X A,REFLO               ; Store TAULO in REFLO
           LD A,[X-]
           X A,REFHI               ; Store TAULO in REFHI

; End of example
```

13.5.2 Application Example: Battery-Powered Weight Measurement

Figure 13-7 shows the block diagram of a simple weight scale application. The pressure sensor circuit is based on a buffered Wheatstone bridge arrangement. A current source and a capacitor generate the linear ramp for the A/D conversion. A crystal oscillator is required for an accurate time base. The modulator is used in 50% duty cycle mode to generate an audible tone. A 24-segment LCD display indicates the weight to the user. Four inputs are used for configuring the scale.

The COP820CJ is held in HALT mode when the appliance is not in use. As soon as a weight is applied to the system, the switch closes, waking up the COP using the Multi-Input Wakeup feature. The same port pin is then reconfigured as an output to power up the sensor circuit, even when the switch is open. Measurement and display are then performed. Finally, the COP820CJ reconfigures the sensor power pin as a pulled-up Wakeup pin, disconnecting the power from the sensor circuit, and then enters HALT mode.

The 16-bit timer is used to generate the interrupts required to refresh the LCD display. A power-on reset circuit (not shown) is required in this application, as the Brown Out should be disabled to keep the HALT mode current as low as possible. With Brown Out disabled, the HALT mode current is typically less than 1uA. The Watchdog circuit is not essential in this application, but could be used to improve system reliability.

13.5.3 Zero Cross Detection

Zero cross detection is often used in appliances connected to the power line. The line frequency is a useful time base for applications such as industrial timers or an iron which switches off if it has not been used for five minutes. Phase-controlled applications require a consistent timing reference in phase with the line voltage.

The COP820CJ requires a square wave, magnitude V_{CC}, at the same frequency as the power line voltage, connected to a input port pin for a simple time base. For a phase-control time base, this waveform should preferably be in phase with the line voltage, although control is still possible if there is a predictable, constant phase lag, less than the phase lag introduced by the load. The choice of zero cross detection circuit depends on factors such as cost, the type of power supply used in the appliance, and the expected interference.

The zero cross detection input can either be polled by software or can be connected to the G0 interrupt line. Polling the pin by software is the simplest technique and saves the interrupt for another function, but has the disadvantage that the polling procedure can be interrupted, causing inaccuracies in synchronization. Disabling the interrupt during the polling is not always possible, as the interrupt may be required for the implementation of other features.

Connecting the zero cross detection input to the external interrupt pin guarantees synchronization. It has the additional advantage that a regular interrupt is generated, which could interrupt the processor out of a fault condition. The interrupt routine only needs to test the integrity of the stack to determine whether such a fault has occurred.

The following software example shows how software polling of the zero cross line is implemented. The application example in Section 13.5.6 shows how interrupt-driven

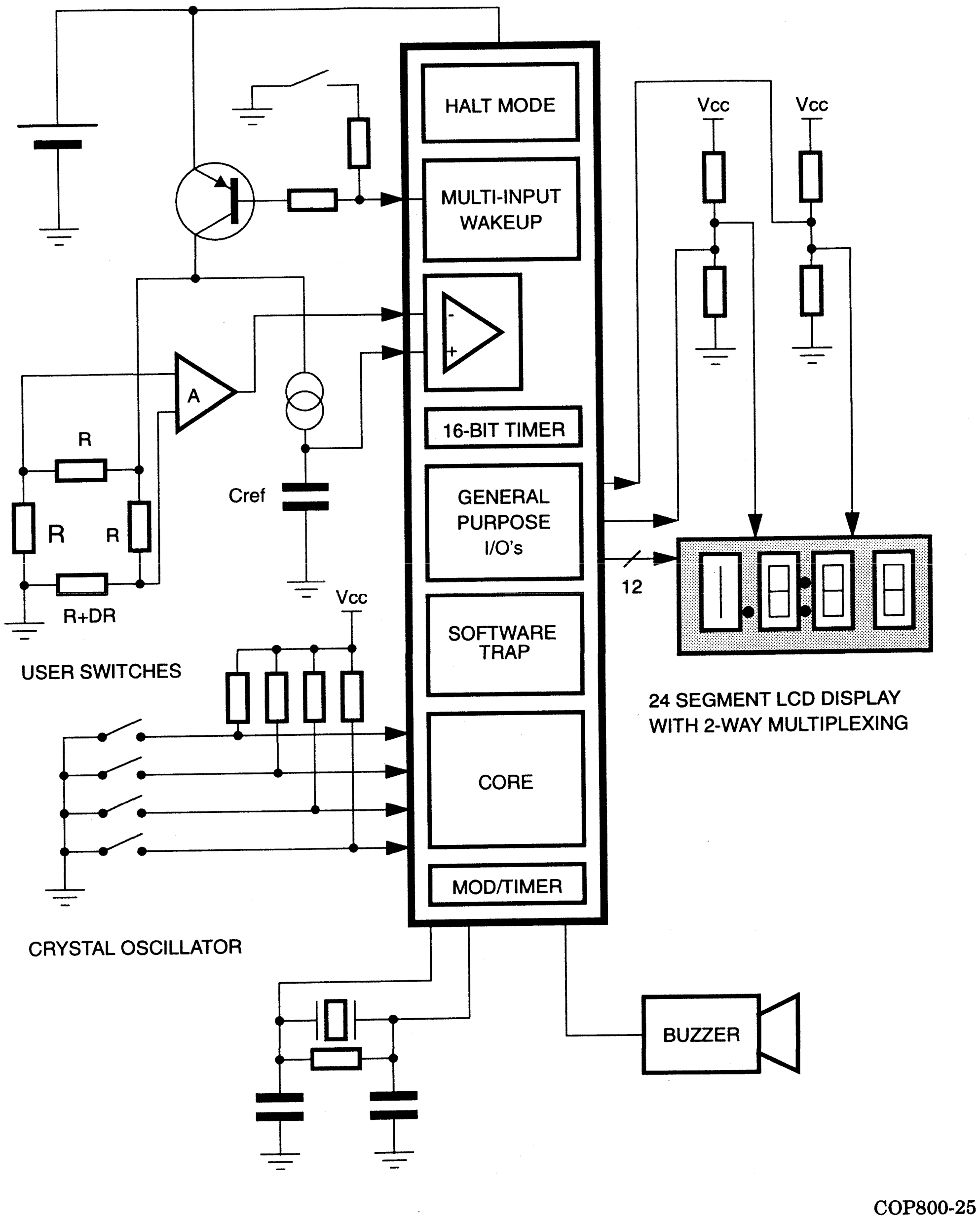


Figure 13-7 Battery-powered Weight Measurement Using COP820CJ

zero cross detection can be used as a time base for phase control of appliances connected to the line.

```
ZCD:
           LD B,#STATUS           ; Save bytes using the B pointer
           IFBIT SYNCHRO,[B]      ; If SYNCHRO is 1, wait for a rising edge
           JP WLOHI               ; otherwise wait for a falling edge.

WHILO:     IFBIT 3,PORTLP         ; Wait for falling edge
           JP WHILO
           SBIT SYNCHRO,[B]       ; SYNCHRO = 1, so wait for rising edge
           JP ENDZCD              ; next time.

WLOHI:     IFBIT 3,PORTLP         ; Wait for a rising edge
           JP RSYNC
           JP WLOHI
RSYNC:     RBIT SYNCHRO,[B]       ; SYNCHRO = 0, so wait for a falling edge
                                  ; next time.

ENDZCD:                           ; End of example
```

13.5.4 Application Example: Industrial Timer

Figure 13-8 shows the block diagram for an industrial timer. The user turns the potentiometer to set the required delay time. When the delay time has elapsed, a load is switched on or off, as selected by the input switches. The time base is derived from the power line using a simple zero cross detection circuit, thereby allowing the use of an inexpensive RC clock instead of a crystal oscillator. There are two indicator diodes and a buzzer driven by the Modulator/Timer.

The A/D conversion routine used by this industrial timer is based on the single slope technique defined in Section 13.5.1, but it has an important difference. Instead of connecting the variable resistor into a voltage divider circuit and measuring the voltage using the single slope technique, the variable resistor forms part of the RC network. The time that the variable RC circuit takes to exceed the fixed reference voltage is directly proportional to the value of the resistor, simplifying the conversion from time into resistance. The circuit as shown can be used to program a time proportional to the angle of the potentiometer setting. The potentiometer can be replaced by a rotary switch connected to a series of resistors, so that each position of the switch generates a different resistance. Here the COP820CJ can identify the switch positions if the difference in each resistance for each position is greater than the inaccuracy in measuring the absolute resistance.

13.5.5 LED Drive Using the COP820CJ

The COP820CJ has four outputs, L4 to L7, which are individually capable of sinking high currents. They are suitable for use in multiplexed, high-efficiency LED displays. Figure 13-9 shows the structure for a three-way multiplexed LED display. Pins L0 to L3 and D0 to D3 drive the LEDs. All the current for the first eight segments is sunk through L4. The current for the second and third set of eight segments is sunk by L5 and L6, respectively. The eight identical resistors connected to the ports and the eight identical resistors connected to the V_{CC} line limit the current. The values of the V_{CC} resistors and

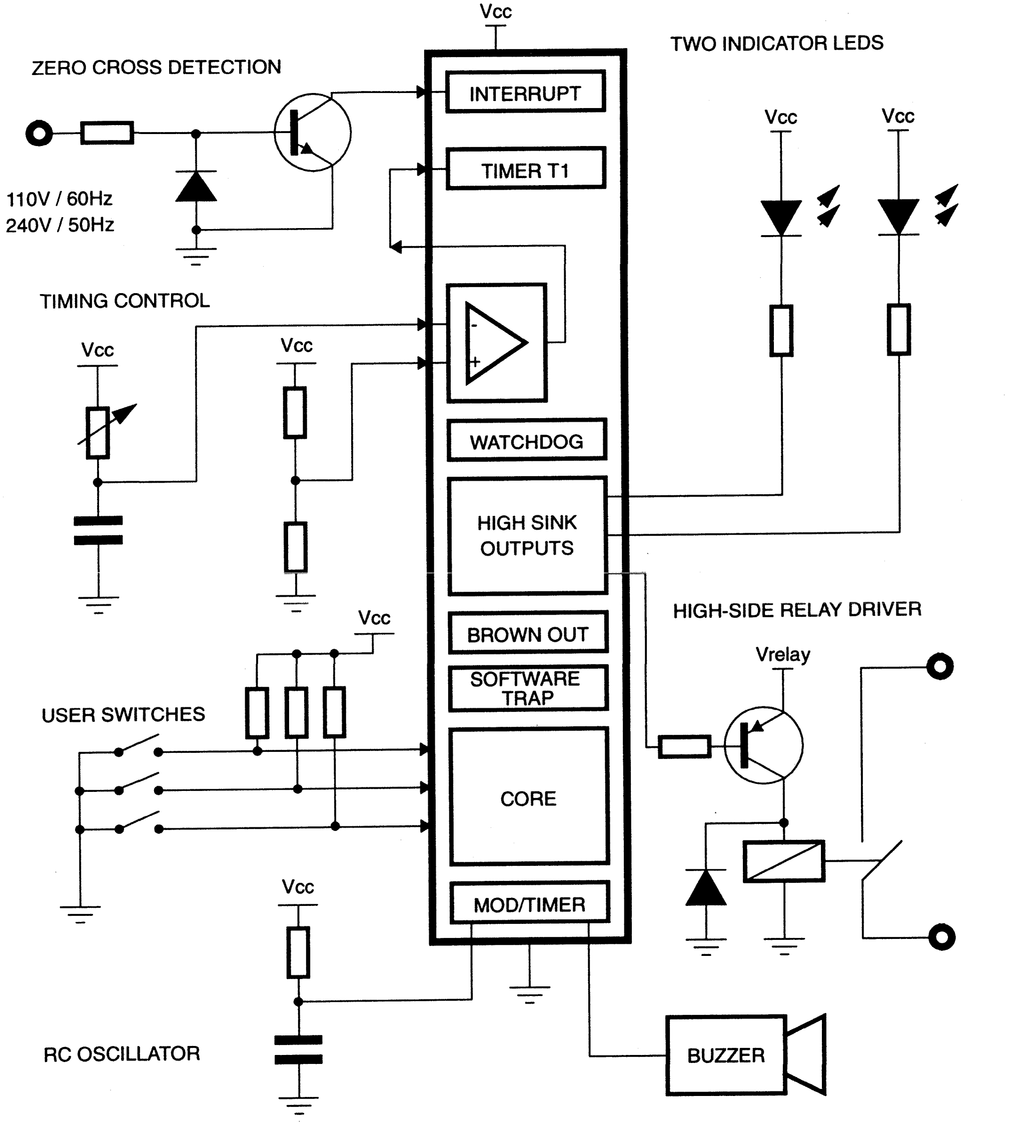
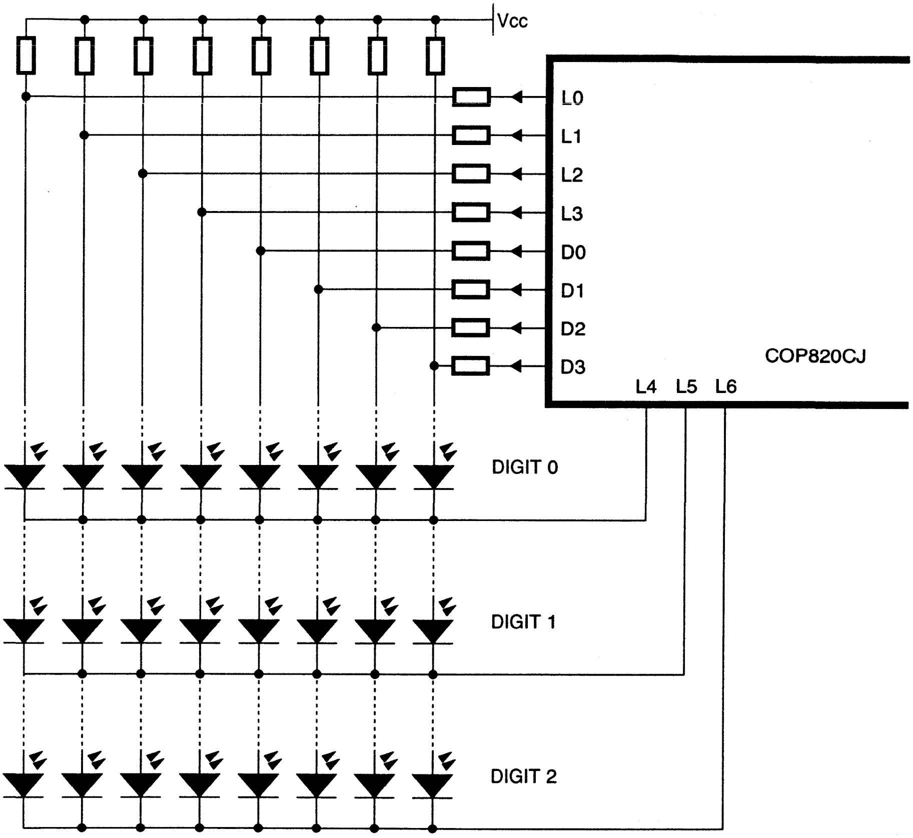


Figure 13-8 Industrial Timer Application Using The COP823CJ



COP800-27

Figure 13-9 3-way Multiplexed Led Display With COP820CJ the port resistors set the current flow into the LED. The ratio of the port resistor value to the V_{CC} resistor value should be sufficiently low so that when the port outputs are switched low, the LED segments are never illuminated.

The multiplexing is performed in the following way. The COP820CJ generates a regular interrupt at a rate known as the multiplex rate. If this rate is too high, the COP will be overloaded. If it is too low, LED flicker will occur. The programmer should set the update rate as required by the application. After the first interrupt, or underflow of the timer if polling the TPND flag is chosen instead, the appropriate bit pattern for the first digit is written to L0-L3 and D0-D3. Pin L4 is set low to enable current to flow through the diodes in the first digit. Pins L5 and L6 are set high to stop current flowing in the second and third digits. The processor waits for the next interrupt or timer underflow, writes the bit pattern of the second digit to the relevant L and D port pins, and sets pin L5 low. L6 and L4 are set high. The third digit is displayed in a similar way, this time setting L6 low and setting L4 and L5 high. The procedure is then repeated.

The following software example demonstrates this procedure. The number in COUNTHI:COUNTLO is initialized to 268 decimal and is displayed on the LEDs. The variable DIGIT is a pointer to the digits.

```
.INCLD COP820CJ.INC

; Variables

DIGIT = 0
COUNTLO = 1
COUNTHI = 2
TEMP = 3

; Start of code

INIT:      LD SP,#02F            ; Initialize stack
           LD PORTLC,#0FF        ; Port L as output
           LD COUNTLO,#068       ; Shown number digit 0 & 1
           LD COUNTHI,#002       ; Shown number digit 2
           LD DIGIT,#0           ; Digit counter
           LD TMRLO,#02F         ; First timer value
           LD TMRHI,#0           ;
           LD TAULO,#0           ;Timer auto reload lo byte
           LD TAUHI,#010         ; Timer auto reload hi byte
           LD CNTRL,#090         ; Start timer, auto reload mode

WAIT:      IFBIT TPND,PSW        ; Timer underflow?
           JP OUT                ; Yes -> OUTPUT
           JP WAIT               ; No -> WAIT

OUT:       RBIT TPND,PSW         ; Reset timer underflow bit
           LD A,#3               ;
           IFEQ A,DIGIT          ; Last digit?
           LD DIGIT,#0           ; Yes -> reset digit counter

DIGXOUT:   JSR DIGOUT            ; Output current digit
           LD A,DIGIT            ; Increment and mask digit counter
           INC A                 ;
           AND A,#03             ;
           X A,DIGIT             ;
           JP WAIT               ;
```

```
.=0100

DIGOUT:    LD A,DIGIT            ; Choose the correct subroutine depending on
           ADD A,#L(TABLE)       ; DIGIT
           JID
TABLE:     ; actual digit table:
           .ADDR DIG0
           .ADDR DIG1
           .ADDR DIG2

DIG0:      LD A,COUNTLO          ; Least significant nibble
           JSR DATA              ; Prepare data lines
           RBIT 4,PORTLD         ; Switch on digit 0
           RET

DIG1:      LD A,COUNTLO          ; Output middle nibble
           SWAP A                ; It’s the higher nibble of COUNTLO
           JSR DATA              ; Prepare data
           RBIT 5,PORTLD         ; Switch on digit 1
           RET

DIG2:      LD A,COUNTHI          ; Output most significant nibble
           JSR DATA              ; Prepare data
           RBIT 6,PORTLD         ; Switch on digit 2
           RET

DATA:
           JSR BCD27             ; Conversion BCD to 7 segment code
           X A,TEMP              ; Save to temporary variable
           LD A,TEMP             ; and restore A
           OR A,#0F0             ; Switch off all digits
           X A,PORTLD            ; and write L Port value
           LD A,TEMP             ; Get actual BCD value
           AND A,#0F0            ; Clear the lower nibble
           X A,TEMP              ; Save value
           LD A,PORTD            ; Read D Port value
           SWAP A                ;
           AND A,#00F            ; Clear the higher nibble
           OR A,TEMP             ; Combine with prepared TEMP
           SWAP A                ;
           X A,PORTD             ; and write it back
           RET
```

```
BCDTAB:                           ; Example of a BCD to 7 segment table
          .BYTE 03F               ; 0
          .BYTE 006               ; 1
          .BYTE 05B               ; 2
          .BYTE 04F               ; 3
          .BYTE 066               ; 4
          .BYTE 06D               ; 5
          .BYTE 07D               ; 6
          .BYTE 007               ; 7
          .BYTE 07F               ; 8
          .BYTE 06F               ; 9
          .BYTE 077               ; A
          .BYTE 07C               ; B
          .BYTE 058               ; C
          .BYTE 05E               ; D
          .BYTE 079               ; E
          .BYTE 071               ; F

BCD27:                            ; BCD to 7 segment conversion routine
          AND A,#00F              ; Mask out upper nibble of A
          ADD A,#L(BCDTAB)        ; Look up value in table.
          LAID
          RET

.END
```

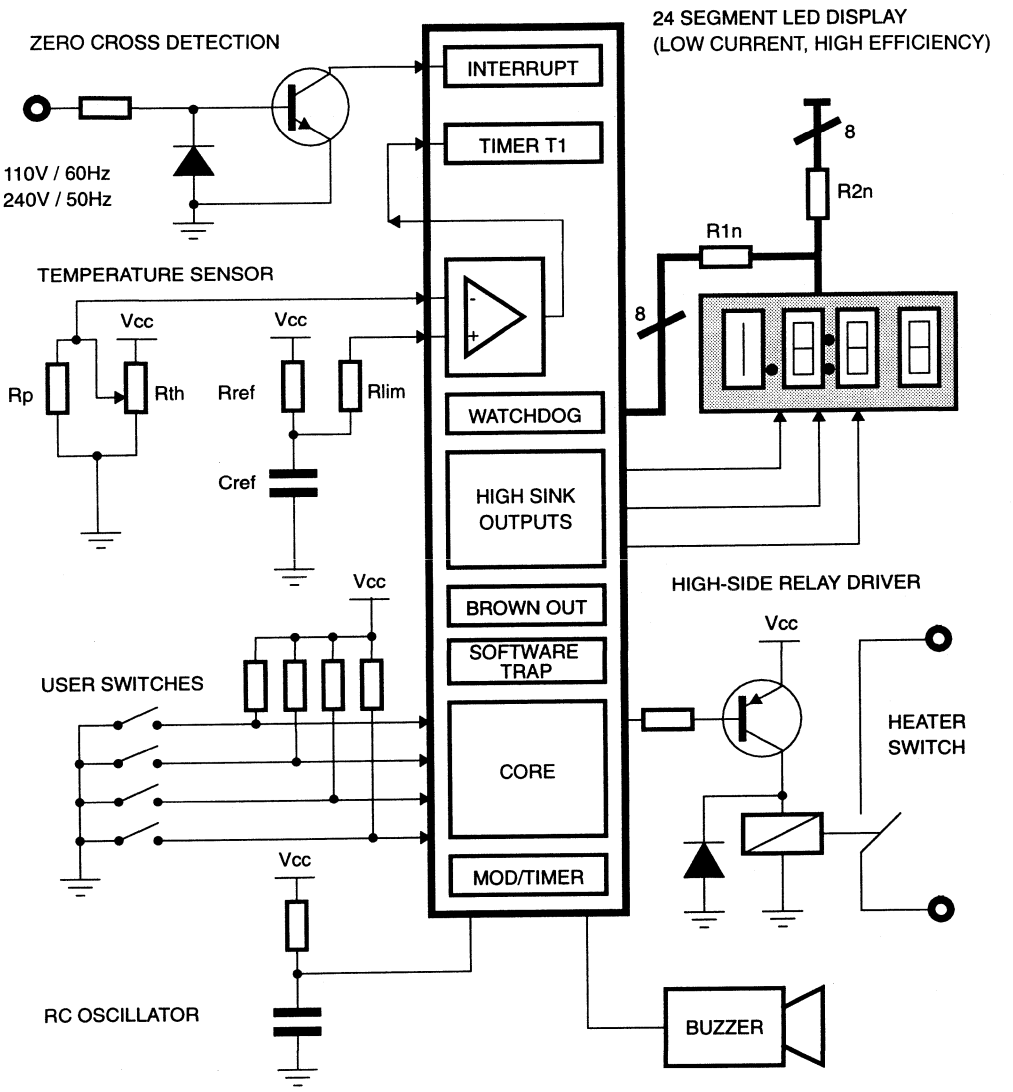
13.5.6 Application Example: Temperature Control

Figure 13-10 shows the block diagram for a household appliance with temperature control such as a coffee maker. The appliance measures the temperature using a thermistor which is linearized with a parallel resistor and connected to the COP820CJ comparator. This configuration performs single slope A/D conversion. The zero cross detection circuit provides the time-base for the system used for calibration of the A/D converter and for the generation of the time display. A high efficiency, low power 24 segment LED display is connected to the COP820CJ to indicate elapsed time and the operating mode to the user. The heater switch is connected to a high-side driver to ensure additional safety. If the relay primary winding is disconnected or shorted to ground, the heater will not operate. Four switches for user input have been provided.

The safety of the system is enhanced by using the Brown Out option, the Watchdog timer and the software interrupt. All unused code areas should be filled with 00 hex, the opcode for the INTR instruction. The Watchdog Timer is used to prevent the program from being caught in an infinite loop. The Brown Out detection protects against transients on the power supply.

13.5.7 Phase Control of an AC Load

The variable duty cycle mode of the Modulator/Timer, in conjunction with a zero cross detection interrupt routine is ideally suited to phase control of single-phase AC loads. The program example below shows how a triac is triggered 6.65 ms after the zero-cross on each half-cycle of the power line. The crystal frequency is assumed to be 10 MHz, resulting in a resolution of 1 µs on Timer T1.



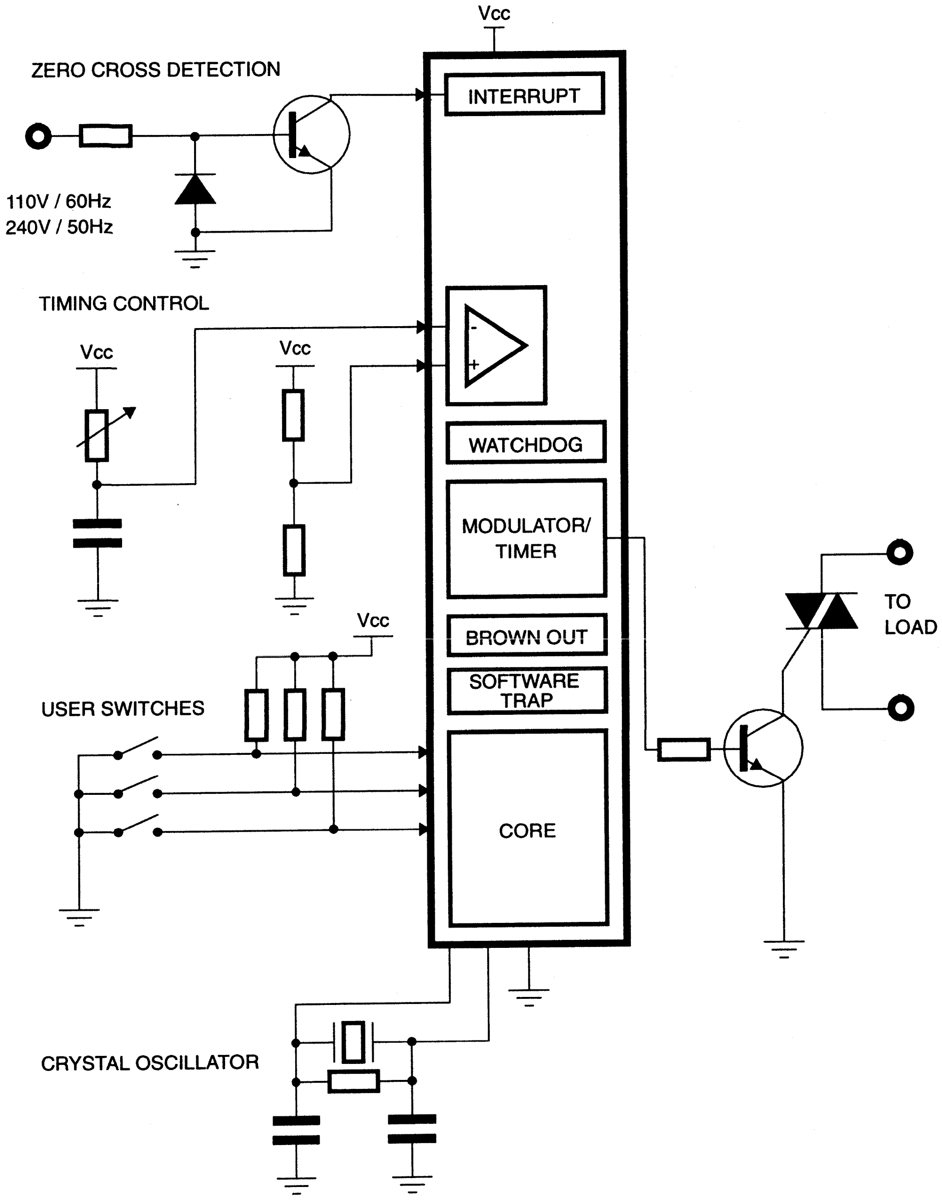
COP800-28

Figure 13-10 Temperature Controlled Appliance Using COP820CJ

After each interrupt, Timer T1 is loaded with the desired angle of 6.655 ms or 01A00 hex and MODRL is loaded with 25 to give a triac gate pulse width of 26 us. The variable duty cycle mode is initialized and Timer T1 is started.

This example is suitable for the phase control in light dimmer or in motor control applications. Figure 13-11 shows a schematic of such an application, using blocks already encountered in the previous examples. Here, the power is directly controlled by the value of the variable resistor. The A/D conversion routine cannot use Timer T1 to generate interrupts in this example. However, Timer T1 can still be read as a time-base within any particular half cycle.

```
.incld COP820CJ.INC

ANGLEL = 000
ANGLEH = 001

START:
           LD SP,#02F            ; Initialize the stack
           RBIT 7,PORTLD         ; Pin L7 is an output, logic low for TRIAC
           SBIT 7,PORTLC
           RBIT 3,PORTGD         ; Pin G3 is a Hi-Z input
           RBIT 3,PORTGC
           RBIT 0,PORTGD         ; Pin G0 is a Hi-Z input for ZCD
           RBIT 0,PORTGC
           LD B,#TAULO
           LD [B+],#0FF          ; Maximum possible value in the
           LD [B+],#0FF          ; auto-reload register
                                 ;
                                 ; B now points to CNTRL
           LD [B+],#0A7          ; Auto-reload, toggle, stop timer, rising
                                 ; interrupt edge
                                 ;
                                 ; B now points to PSW
            LD [B],#002          ; External ZCD interrupts enabled,
                                 ; pending flags cleared

           LD B,#ANGLEL
           LD [B+],#000          ; Set the firing angle to 1A00 hex
           LD [B+],#01A

           SBIT GIE,PSW          ; Enable ZCD interrupts

WAIT:      JP WAIT               ; Wait for interrupt

; Interrupt handler

.=0FF

IHDL:
           IFBIT IPND,PSW        ; An external interrupt indicates a zero
           JP ZCD                ; cross event.

FAIL:      JP FAIL               ; A software interrupt will have
                                 ; caused this event.

ZCD:       RBIT TRUN,CNTRL1      ; Stop Timer T1
```



NOTE: Either V_{CC} or GND must be connected to one of the power terminals for this to function.

COP800-29

Figure 13-11 AC Phase Control Application Using COP820CJ

```
          LD B,#ANGLEL
          LD X,#TMRLO
          LD A,[B+]              ; Load Timer T1 with the desired angle value
          X A,[X+]
          LD A,[B+]
          X A,[X+]

          LD MODRL,#25           ; Set the TRIAC firing pulse width to 26us

          LD CNTRL2,#040         ; Set up modulator timer into variable
                                 ; duty cycle mode

          RBIT 7,PORTLD          ; Set up pin L7 to output logic 0

          SBIT TRUN,CNTRL1       ; Start the timer.

EDGESW:
          LD B,#CNTRL1
                                 ; Toggle the interrupt routine edge so
                                 ; that both +ve and -ve half cycles are
          IFBIT IEDG,[B]         ; used.
          JP REDG
          SBIT IEDG,[B
          JP ENDINT
 REDG:    RBIT IEDG,[B]

ENDINT:
          RBIT IPND,PSW
          RETI

; End of the example
```

13.5.8 Application Example: Remote Control Unit

A battery-powered remote control unit application using the COP820CJ is presented in Chapter 11. The unit transmits a specific code using an infrared LED each time a particular key is pressed. For details, see Chapter 11.

13.6 PROGRAMMING EXAMPLES

This section is intended to be an overview of programming examples. For more detailed and varied programming examples, refer to the Microcontroller Databook or the Microcontroller Applications Engineering BBS at (408) 739-1162 (8NI).

13.6.1 Clear RAM

The following program clears all RAM locations except for the stack pointer. The value of the argument to IFBNE may need to be adjusted, depending on the size of RAM in specific family members.

COP800 PROGRAM TO CLEAR ALL RAM EXCEPT SP

```
addr:
0000:    LD       0FC,#070 ;Define X-pointer as counter
0002:    LD       B,#0     :Initialize B pointer
0003:    LD       [B+],#0  ;Load mem with 0 and incr B pointer
0005:    DRSZ     0FC      ;Decrement counter
0006:    JP       0003     ;Skip if lower half RAM is cleared
0007:    LD       B,#0F0   ;Point B to upper half of RAM
0009:    LD       [B+],#0  ;Load upper RAM half with 0
000B:    IFBNE    #0D      ;until B points to 0FD (=SP)
000C:    JP       009      ;Skip if B=0FD
000D:    LD       B,#0     ;Initialize B to 0
```

13.6.2 Binary/BCD Arithmetic Operations

The arithmetic instructions include the Add (ADD), Add with Carry (ADC), Subtract with Carry (SUBC), Increment (INC), Decrement (DEC), Decimal Correct (DCOR), Clear Accumulator (CLR), Set Carry (SC), and Reset Carry (RC). The shift and rotate instructions, which include the Rotate Right through Carry (RRC), and the Swap accumulator nibbles (SWAP), may also be considered arithmetic instruction variations. The RRC instruction is instrumental in writing a fast multiply routine.

In subtraction, a borrow is represented by the absence of a Carry and vice versa. Consequently, the Carry flag needs to be set (no borrow) before a subtraction, just as the Carry flag is reset (no carry) before an addition. The ADD instruction does not use the Carry flag as an input. It should also be noted that both the Carry and Half Carry flags (Bits 6 and 7, respectively, of the PSW control register) are cleared with RESET and remain unchanged with the ADD, INC, DEC, DCOR, CLR, and SWAP instructions. The DCOR instruction uses both the Carry and Half Carry flags. The SC instruction sets both the Carry and Half Carry flags, while the RC instruction resets both these flags.

The following program examples illustrate additions and subtractions of 4-byte data fields in both binary and BCD (Binary Coded Decimal). The four bytes from data memory locations 24 through 27 are added to or subtracted from the four bytes in data memory locations 16 through 19. The results replace the data in memory locations 24 through 27.

These operations are performed both in binary and BCD. It should be noted that the BCD preconditioning of adding (ADD) the hexadecimal value 66 is necessary only with the BCD addition, not with the BCD subtraction. The binary coded decimal DCOR (Decimal Correct) instruction uses both the Cary and Half Carry flags as inputs but does not change the Carry and Half Carry flags. Also note that the #12 with the IFBNE instruction represents 28 minus 16, since the IFBNE operand is modulo 16 (remainder when divided by 16).

BINARY ADDITION:

```
         LD      X,#16     ;NO LEADING ZERO
         LD      B,#24     ;INDICATES DECIMAL
         RC
LOOP:    LD      A,[X+]
         ADC     A,[B]
         X       A,[B+]
         IFBNE   #12
         JP      LOOP
         IFC
         JP      OVFLOW    ;OVERLFOW IF C
```

BINARY SUBTRACTION:

```
         LD      X,#010    ;LEADING ZERO
         LD      B,#018    ;INDICATES HEX
         SC
LOOP:    LD      A,[X+]
         SUBC    A,[B]
         X       A,[B+]
         IFBNE   #12
         JP      LOOP
         IFNC
         JP      NEGRSLT   ;NEG. RESULT IF NO C
                           (NO C = BORROW)
```

BCD ADDITION:

```
         LD      X,#010    ;LEADING ZERO
         LD      B,#018    ;INDICATES HEX
         RC
LOOP:    LD      A,[X+]
         ADD     A,#066    ;ADD HEX 66
         ADC     A,[B]
         DCOR    A         ;DECIMAL CORRECT
         X       A,[B+]
         IFBNE   #12
         JP      LOOP
         IFC
         JP      OVFLOW    ;OVERFLOW IF C
```

BCD SUBTRACTION:

```
         LD      X,#16     ;NO LEADING ZERO
         LD      B,#24     ;INDICATES DECIMAL
         SC
LOOP:    LD      A,[X+]
         SUBC    A,[B]
         DCOR    A         ;DECIMAL CORRECT
         X       A,[B+]
         IFBNE   #12
         JP      LOOP
         IFNC
         JP      NEGRSLT   ;NEG. RESULT IF NO C
                           (NO C = BORROW)
```

Note that the previous additions and subtractions are not “adding machine” type arithmetic operations in that the result replaces the second operand rather that the first. The following program examples illustrate “adding machine” type operations where the result replaces the first operand. With subtraction, this entails the result replacing the minuend rather that the subtrahend.

```
BINARY ADDITION:

         LD       B,#16
         LD       X,#24
         RC
LOOP:    LD       A,[X+]
         ADC      A,[B]
         X        A,[B+]
         IFBNE    #4
         JP       LOOP
         IFC
         JP       OVFLOW    ;OVERLFOW IF C
```

```
BINARY SUBTRACTION:

         LD       B,#010
         LD       X,#018
         SC
LOOP:    LD       A,[X+]
         X        A,[B]
         SUBC     A,[B]
         X        A,[B+]
         IFBNE    #4
         JP       LOOP
         IFNC
         JP       NEGRSLT   ;NEG. RESULT IF NO C
                            (NO C = BORROW)
```

```
BCD ADDITION:

         LD       B,#010
         LD       X,#018
         RC
LOOP:    LD       A,[X+]
         ADD      A,#066
         ADC      A,[B]
         DCOR     A
         X        A,[B+]
         IFBNE    #4
         JP       LOOP
         IFC
         JP       OVFLOW    ;OVERFLOW IF C
```

```
BCD SUBTRACTION:

         LD       B,#16
         LD       X,#24
         SC
LOOP:    LD       A,[X+]
         X        A,[B]
         SUBC     A,[B]
         DCOR     A
         X        A,[B+]
         IFBNE    #4
         JP       LOOP
         IFNC
         JP       NEGRSLT   ;NEG. RESULT IF NO C
                            (NO C = BORROW)
```

The following hybrid arithmetic example adds five successive bytes of a data table in ROM program memory to a two-byte SUM, and then subtracts the SUM from a two-byte total TOT. Assume that the ROM table is located starting a program memory address 0401, while SUM and TOT are at RAM data memory locations 1, 0 and 3, 2, respectively, and that the program is encoded as a subroutine.

```
ROM TABLE:
         .=0401
         .BYTE     102
         .BYTE     41
         .BYTE     31
         .BYTE     26
         .BYTE     5
TABLE: TOP DOWN
ARTHMETIC: BOTTOM UP
         SUMLO = 0
         SUMHI = 1
         TOTLO = 2
         TOTHI = 3
ARITH1:  LD        X,#5      ;SET UP ROM TABLE PTR
         LD        B,#0      ;SET UP SUM POINTER
LOOP:    RC
         LD        A,X       ;LOAD ROM PTR INTO ACC
         LAID                ;READ VALUE FROM ROM
         ADC       A,[B]     ;ADD SUMLO TO ROM VALUE
         X         A,[B+]    ;PUT RSLT BACK IN SUMLO
         CLR       A         ;CLR ACC
         ADC       A,[B]     ;ADD SUMHI TO ACC
         X         A,[B-]    ;PUT RSLT BACK IN SUMHI
         DRSZ      X         ;DECR. & TEST ROM PTR
         JP        LOOP      ;REPEAT LOOP IF PTR NOT 0
         SC
         LD        B,#2
LUP:     LD        A,[X+]    ;LOAD SUBTRAHEND FIRST
         X         A,[B]     ;REVERSE OPERANDS FOR SUBTRACTION
         SUBC      A,[B]
         X         A,[B+]    ;INCR. MINUEND POINTER
         IFBNE     #4        ;REPEAT LOOP IF B PTR NOT EQUAL TO 4
         JP        LUP
         RET
```

13.6.3 Binary Multiplication

The following program listing shows the program for a 16-by-16-bit binary multiply subroutine. The multiplier starts in the lower 16 bits of the 32-bit result location. As the multiplier is shifted out of the low end of the result location with the RRC instruction, each multiplier bit is tested in the Carry flag. The multiplicand is conditionally added (depending on the multiplier bit) into the high end of the result location, after which the partial product is shifted down one bit position following the multiplier. Note that one additional terminal shift cycle is necessary to align the result.

```
          ;COP800   MULTIPLY (16X16) SUBROUTINE
          ;         MULTIPLICAND IN [1,0] MULTIPLIER IN [3,2]
          ;         PRODUCT IN [5, 4, 3, 2]
          ;
          ;         VERNE H. WILSON 11/7/86
          ;
     00F0           CNTR = 0F0
04B7 D011 MULT:     LD        CNTR,#17
04B9 5B             LD        B,#4
04BA 9A00           LD        [B+1],#0
04BC 9E00           LD        [B],#0
04BE DC00           LD        X,#0
04C0 A0             RC
04C1 AE   MLOOP:    LD        A,[B]
04C2 B0             RRC       A
04C3 A3             X         A,[B-]
04C4 AE             LD        A,[B]
04C5 B0             RRC       A
04C6 A3             X         A,[B-]
04C7 AE             LD        A,[B]
04C8 B0             RRC       A
94C9 A3             X         A,[B-]
04CA AE             LD        A,[B]
04CB B0             RRC       A
04CC A6             X         A,[B]
04CD 5A             LD        B,#5
04CE 89             IFNC
04CF 08             JP        TEST
04D0 A0             RC
04D1 5B             LD        B,#4
04D2 BA             LD        A,[X+]
04D3 80             ADC       A,[B]
04D4 A2             X         A,[B+]
04D5 BB             LD        A,[X-]
04D6 80             ADC       A,[B]
04D7 A6             X         A,[B]
04D8 C0   TEST:     DRSZ      CNTR
04D9 E7             JP        MLOOP
04DA 8E             RET
```

13.6.4 Binary Division

The following program shows a subroutine for a 16-by-16-bit binary division. A 16-bit quotient is generated along with a 16-bit remainder. The dividend is left shifted up into an initially-cleared 16-bit test window where the divisor is test-subtracted. If the test subtraction generates no high-order borrow, then the real subtraction is performed with the result stored back in the test window. At the same time, a quotient bit (equal to 1) is inserted into the low end of the dividend window to record that a real subtraction has taken place. The entire dividend and test window is then shifted up (left shifted) one bit position with the quotient following the dividend.

Note that the four left shifts (LD, ADC, X) in the LSHFT section of the program are repeated as straight-line code rather than a loop in order to optimize throughput time.

```
COP800   DIVIDE (16X16) SUBROUTINE
         DIVIDEND IN [3,2]
         DIVISOR IN [1,0]
         QUOTIENT IN [3,2]
         REMAINDER IN [5,4]

         CNTR = 0F0

DIV:     LD          CNTR,#16
         LD          B,#5
         LD          [B-],#0
         LD          [B],#0
         LD          X,#4
LSHFT:   RC
         LD          B,#2
         LD          A,[B]
         ADC         A,[B]
         X           A,[B+]
         LD          A,[B]
         ADC         A,[B]
         X           A,[B+]
         LD          A,[B]
         ADC         A,[B]
         X           A,[B+]
         LD          A,[B]
         ADC         A,[B]
         X           A,[B+]
TSUBT:   SC
         LD          B,#0
         LD          A,[X+]
         SUBC        A,[B]
         LD          B,#1
         LD          A,[X-]
         SUBC        A,[B]
         IFNC
         JP          TEST
SUBT:    LD          B,#0
         LD          A,[X]
         SUBC        A,[B]
         X           A,[X+]
         LD          B,#1
         LD          A,[X]
         SUBC        A,[B]
         X           A,[X-]
         LD          B,#2
         SBIT        0,[B]
TEST:    DRSZ        CNTR
         JMP         LSHIFT
         RET
```

With a division where the dividend is larger than the divisor (relative to the number of bytes), an additional test step must be added. This test determines whether a high-order carry is generated from the left shift of the dividend through the test window. When this carry occurs, the program branches directly to the SUBT subtract routine. This carry can occur only if the divisor contains a high-order bit. Moreover, the divisor must also be larger than the shifted dividend when the shift has placed a high-order bit in the test window. When this case occurs, the TSUBT test subtract shows the divisor to be larger than the shifted dividend and no real subtraction occurs. Consequently, the high-order bit of the shifted dividend is again left shifted and results in a high-order carry. This test is illustrated in the following program for a 24-by-8-bit binary division.

Note that the four left shifts (LD, ADC, X) in the LSHFT section of the program are repeated with the JP jump to LUP instruction in order to minimize program size.

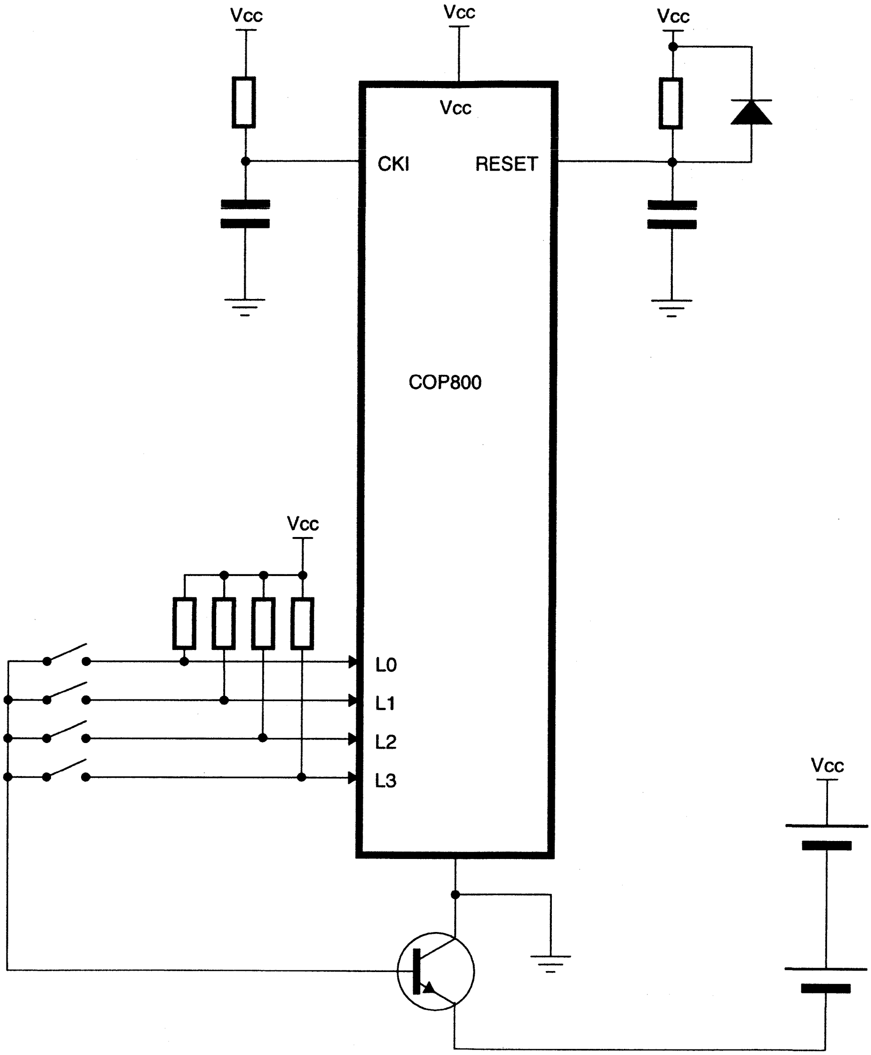
```
COP800   DIVIDE (24X8) SUBROUTINE
         DIVIDEND IN [2,1,0]
         DIVISOR IN [4]
         QUOTIENT IN [2,1,0]
         REMAINDER IN [3]

         CNTR = 0F0

DIV:     LD          CNTR,#24
         LD          B,#3
         LD          [B],#0
LSHFT:   RC
         LD          B,#0
LUP:     LD          A,[B]
         ADC         A,[B]
         X           A,[B+]
         IFBNE       #4
         JP          LUP
         IFC
         JP          SUBT
TSUBT:   SC
         LD          B,#3
         LD          A,[B+]
         SUBC        A,[B]
         IFNC
         JP          TEST
SUBT:    LD          A,[B-]
         X           A,[B]
         SUBC        A,[B]
         X           A,[B]
         LD          B,#0
         SBIT        0,[B]
TEST:    DRSZ        CNTR
```

13.7 EXTERNAL POWER WAKEUP CIRCUIT

Power-on wakeup is a technique used in battery powered applications such as electronic keys or digital scales to save battery power. Instead of using the HALT mode when the application is not in use, the COP device is powered off. If there is only one input switch in the application, the implementation is simple. This switch is put in series with the battery, providing power to the circuit when the switch is closed.

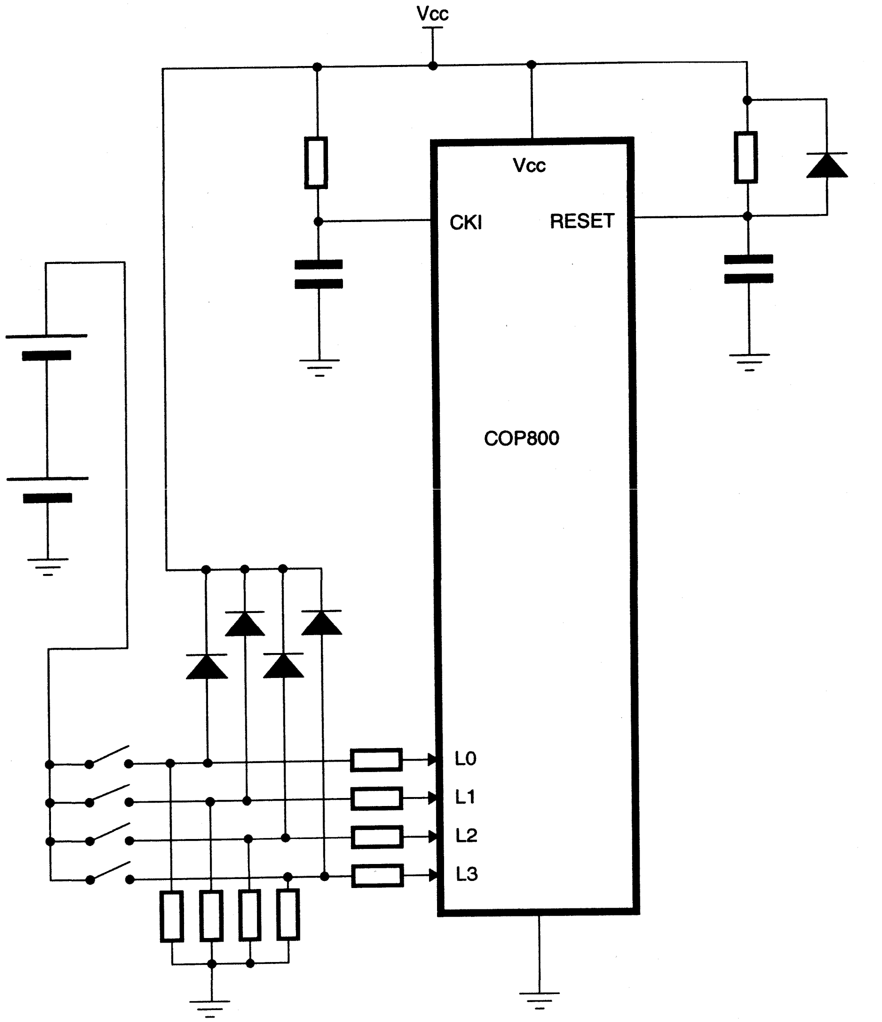
If there is more than one switch, power-on wakeup can be achieved by using an NPN transistor and one resistor per switch as shown in Figure 13-12. Here, the circuit ground is connected to the battery negative terminal via the NPN transistor. If the base is floating, it will not conduct. If the base is pulled to V_{CC} via a current-limiting resistor, it will conduct, powering up the circuit.

An alternative technique is shown in Figure 13-7. Here the positive terminal of the battery is connected to the V_{CC} line via a switch, a diode and two resistors per line. If a switch is pressed, power is applied to the V_{CC} line. The pull-down resistors pull any ports connected to open switches to ground. If the switch is closed, the voltage on the switch will be V_{CC} plus the diode voltage drop. If this potential were directly applied to the L port pin, the COP device would be driven outside the operating specification. Therefore, series protection resistors are used on all Port L pins connected to the switches.



COP800-30

Figure 13-12 Power Wakeup Using An NPN Transistor

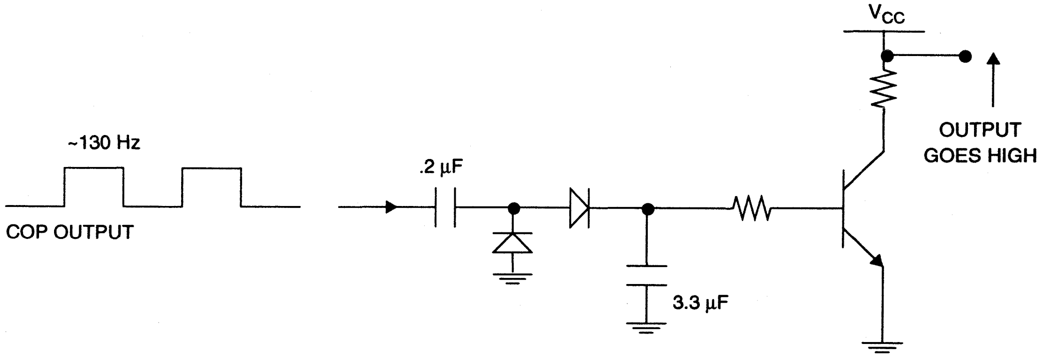


COP800-31

Figure 13-13 Power Wakeup Using Diodes And Resistors

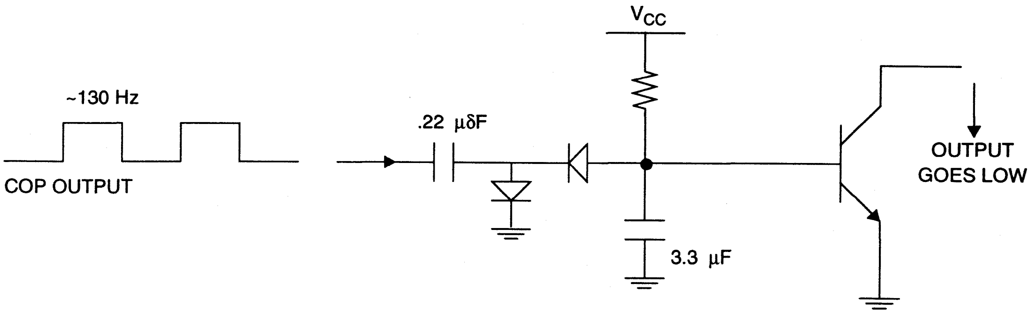
13.8 EXTERNAL WATCHDOG CIRCUIT

In the following application examples, the COPS device sends a continuous square wave to an external Watchdog circuit. If the user program gets stuck in a software loop and the square wave is not generated, the external circuit will provide a high transition (Circuit A) or a low transition (Circuit B). The output of the Watchdog circuit may be connected to the COP $\overline{\text{RESET}}$ pin or the system reset in order to generate a reset on a Watchdog error.



COP800-35

Figure 13-14 External Watchdog Circuit A

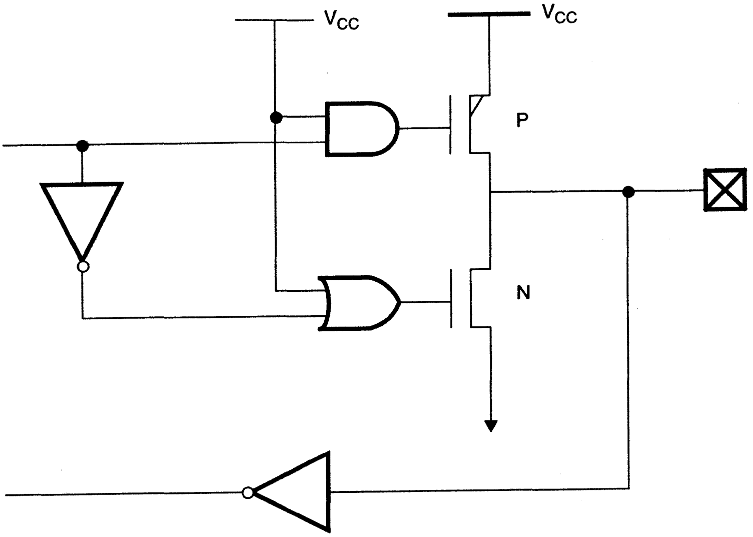


COP800-36

Figure 13-15 External Watchdog Circuit B

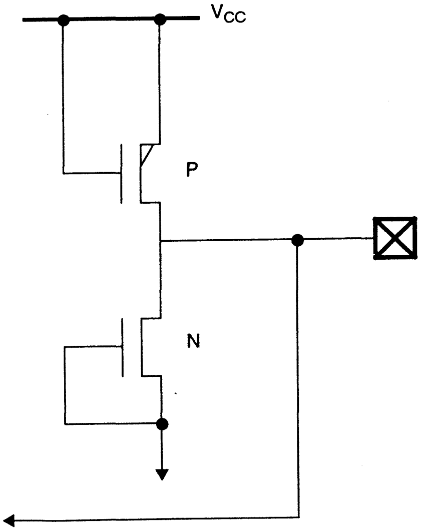
13.9 INPUT PROTECTION ON COP800 PINS

The COP800 input pins have internal circuitry for protection from ESD. The internal circuitry is shown in Figures 13-16 and 13-17.



COP800-37

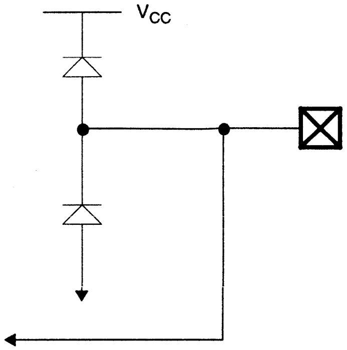
Figure 13-16 Ports L/C/G Input Protection (Except G6)



COP800-38

Figure 13-17 Port I Input Protection

The input protection circuitry is implemented with the P_channel transistors. The equivalent diode circuit is shown in Figure 13-18.



COP800-39

Figure 13-18 Diode Equivalent of Input Protection

When the inputs are tri-stated and the input voltage on the pin is between GND and V_{CC}, the input protection diodes are off. The only current drawn into or out of the pin is leakage current. If the input is expected to be below GND and/or above V_{CC}, an external series resistor must be used to limit the input current below the maximum allowable current.

In addition to limiting the input current to below the maximum latchup spec (specified in the datasheet), the user should also consider the fact that drawing excessive continuous current into the pin, even though below the maximum latchup current, may cause overstress.

A typical example of drawing continuous current is in an automotive application where the ignition signal (battery) is connected to an input pin through a series resistor. Assuming a 100K series resistor with a tolerance of ±10%, the worst case resistor value is 90K. The battery voltage is assumed to be 12V for normal operation and 24V for a "jump start." The high voltage applied to the pin causes the on-chip protection diode to be forward biased, resulting in current into the associated V_{CC} metal trace. Based on a diode threshold voltage of 0.6V, the voltage at the pin will be V_{CC} + 0.6V. Based on a V_{CC} value of 5V, the input current can be calculated as follows:

Normal Operation:

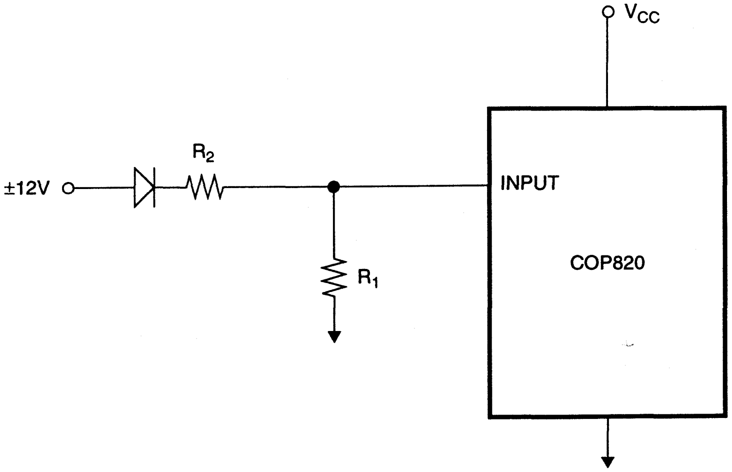
$$\text{Input current} = \frac{[12-(5+0.6)]}{90'\text{K}} = 71\mu'\text{A}$$

Jump Start:

$$\text{Input current} = \frac{[24+(5+0.6)]}{90'\text{K}} = 204\mu'\text{A}$$

A study of the internal circuitry indicates that the input pin can draw about 200 µA without causing any damage or reliability problem.

Another approach is to use appropriate external circuitry that prevents the input protection diodes from being biased. An example is shown in Figure 13-19.



COP800-40

Figure 13-19 External Protection of Inputs

The resistors are required to drop the +12V and the diode prevents the –12V from being applied to the pin.

For V_I = 12V ±5% and V_{CC} = 5V ± 5%, the resistor values are calculated to be:

R_1 = 47K +5%

R_2 = 82K ±5%

This analysis does not apply to G6, $\overline{\text{RESET}}$, and CKI which do not have the protection diodes. Implementation of the above circuit will result in a V_{IH} that is between 0.7 V_{CC} and V_{CC}, and a V_{IL} that is between V_{SS} (0V) and 0.2 V_{CC}.

13.10 ELECTROMAGNETIC INTERFERENCE (EMI) CONSIDERATIONS

13.10.1 Introduction

CMOS has become the technology of choice for the processors used in many embedded systems due to its capability for low standby power consumption. However, CMOS is prone to high current transients on the power supply as the internal logic switches. These transients can easily be the source of high-frequency emissions from the system. The system designer should anticipate and minimize unwanted electromagnetic interference (EMI).

13.10.2 Emission Predictions

"EMI in a typical electronic circuit is generated by a current flowing in a loop configured within the circuit. These paths can be either V_{CC}-to-GND loops or output-to-GND loops. EMI generation is a function of several factors. Transmitted signal frequency, duty cycle, edge rates, and output voltage swings are the major factors of the resultant EMI levels."[1]

The formula for predicting the Electric Field emissions from such a loop is as follows:

$$|E|_{MAX} = \frac{1.32 \times 10^{-3} IA\,(Freq)^2}{D} \times \left(1 + \left(\frac{\lambda}{2\pi D}\right)^2\right)^{1/2}$$

where:

- $|E|_{MAX}$ is the maximum E-field in the plane of the loop in µV/m
- I is the current amplitude in milliamps
- A is the loop area in square cm
- λ is the wavelength at the frequency of interest in meters
- D is the observation distance in meters
- $Freq$ is the frequency in MHz
- and the perimeter of the loop $P \ll \lambda$.

Applying this equation to a single standard output for a National Semiconductor Microcontroller, and performing a Fourier analysis of the output switching at a frequency of 20 MHz, yields the results shown in Table 13-1. These calculations assume a trace length of 5 inches, a board thickness of 0.062 inches and a full ground plane. The load capacitance is 100 pf.

Table 13-1 Electric Field Calculation Results

Harmonic (MHz)	**Current (mA)**	**$\|E\|_{Max}$ (µV/M)**	**$\|E\|_{Max}$ (dBµV/M)**
20	37.56	8.3	18.4
40	3.66	0.3	-10.2
60	26.13	44.2	33.0
80	4.44	0.6	-4.4
100	16.82	80.2	38.1
120	4.71	2.0	6.0
140	11.21	104.0	40.4
160	4.86	5.8	15.2
180	7.82	127.4	42.1

[1] "FACT™ Advanced CMOS Logic Databook", National Semiconductor, 1989

Note that the assumption is made that the output is switching at 20 MHz, which is rarely the case for a port output. There is noise, however, on the output at these frequencies due to switching within the device. This is the noise which is coupled to the output through V_{CC} and GND. Another point to keep in mind is that rarely does one single output switch, but usually several at one time, thus adding the effective magnetic fields from all the outputs which are switching.

Accurate analysis requires characterization of the noise present at the output due to V_{CC} or GND noise which is dependent on many factors, including internal peripherals in use, execution code, and address of memory locations in use.

13.10.3 Board Layout

There are two primary techniques of reducing emissions from within the application. This can be done either by reducing the noise or by controlling the antenna. Control of the antenna is accomplished through careful PC board layout.

General

Standard good PC layout practices will go a long way toward reducing emissions. Traces carrying large AC currents (such as signals with fast transition times, that drive large loads) should be kept as short as possible. Traces that are sensitive to noise should be surrounded by ground to the greatest extent possible. Ground and V_{CC} traces should be kept as short and wide as possible to reduce the supply impedance.

Ground Plane

One of the most effective ways to control emissions through board layout is with a ground plane. The use of a plane can help by providing a return path for fast switching signals, thus reducing loop size for both power and signals.

Multilayer Board

The best way to provide a ground plane is through the use of a multilayer printed circuit board. The large area and the proximity of the V_{CC} and GND planes provide additional decoupling for the power, and provide effective return paths for both power and signals.

The problem with the use of a multilayer board, particularly in consumer related industries, is cost. Due to the volumes involved, an addition of several dollars to the cost of an item may be prohibitive.

13.10.4 Decoupling

Control of the emitted noise can be accomplished by several techniques, including decoupling, reduced power supplies, and limitation of signal strength by the addition of series resistance.

It is important to take the time to properly design the decoupling for CMOS processors. Two decoupling techniques can and should be used to minimize both voltage and current switching noise in the system.

Capacitive Decoupling

Capacitive decoupling is commonly used to control voltage noise on the V_{CC} and GND lines of the board, but if the decoupling is properly designed and is kept as close as possible to the power pins of the device, it can also reduce the effective loop area and thus the antenna efficiency. Capacitive decoupling can prevent high-frequency current transients from being seen by the power supply.

One factor of capacitive decoupling which is often overlooked is the frequency response of the capacitors. Each capacitor, dependent on value, lead length, and dielectric material, possesses a series resonant frequency beyond which the device has inductive characteristics. This inductance inhibits the capacitor from responding quickly to the current needs of the processor and forces the current to use the longer path back to the main power supply.

These inductive characteristics can be countered by the addition of extra capacitors of different values in parallel with the original device. As the value of the capacitor decreases (for capacitors of similar manufacture), the resonant frequency increases.

Placing multiple decoupling capacitors across the power pins of the processor can effectively improve the high frequency performance of the decoupling network. Capacitance values are normally selected which are separated by a decade. However, it is best to check the specifications of the capacitors which are used.

Inductive Decoupling

Another very effective method of decoupling which is rarely used is inductive decoupling. The proper placement of ferrite beads between the decoupling capacitors and the processor can significantly reduce the current noise on the power pins.

The use of inductive decoupling, which will increase the series impedance of the power supply, appears to be contradictory to the effect of capacitive decoupling. However, the purpose of inductive decoupling is to force nodes internal to the processor, which are not switching, into providing the charge for the nodes which are switching.

Ferrite beads are very effective for this type of decoupling due to their lossy nature. Rather than storing the energy and returning it to the circuit later, ferrites will dissipate the energy as a resistor.

One should be aware of potential repercussions from the use of any type of series isolation from the power supply. Due to the reduced V_{CC} which may be present during switching transients, interfacing to other devices in the system may be a problem. Since the V_{CC} should only be reduced for the duration of the switching transient, this should only be a problem if the other devices have especially sensitive and fast-responding inputs.

13.10.5 Output Series Resistance

The addition of resistance in series with outputs can have a significant effect on the emissions caused by the switching of the outputs.

Outputs that drive large capacitive loads can have a lot of current flowing when they switch. While the series resistance may slow the switching speed of the node and thus affect the propagation delay, it can also have a large effect on emissions by reducing the amplitude of the current spike that charges or discharges the load.

13.10.6 Oscillator Control

One very definite source of emissions is the system clock. The some oscillator is intended to switch at high speed and therefore will emit some noise. Keeping the circuit loop of the oscillator as small as possible will help considerably.

Ceramic resonators are available with the capacitive load included in a single three terminal package. The use of these devices and placing them right next to the processor can reduce emissions as much as 10 dB.

RC oscillators are particularly troublesome for emissions due to the high transient current when the processor turns on the N-channel device that discharges the capacitor. The transistor is meant to be large and to turn on strongly in order to discharge the capacitor as quickly as possible. This allows simple control over the frequency of oscillation but causes difficulty for the designer of systems for EMI-sensitive applications.

13.10.7 Mechanical Shielding

A last resort for controlling emissions is the addition of mechanical shielding. While shielding can be effective and can be easier from an electrical design standpoint, the implementation and installation of a proper electromagnetic shield can be excessively costly and time consuming.

It is much better to design the system with the control of emissions in mind from the start rather than to apply bandages when it is time to begin production.

13.10.8 Conclusion

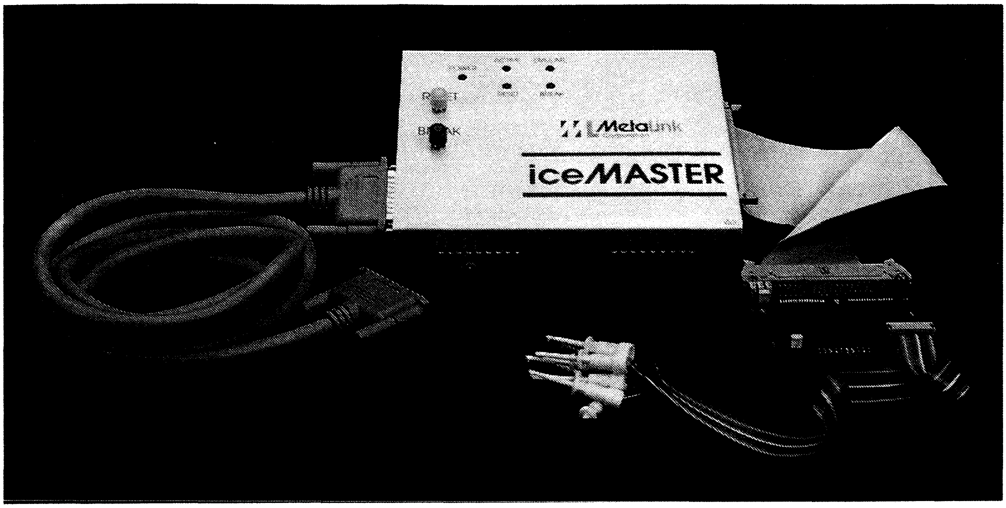
While electromagnetic emissions can be a problem for the designer of any electronic system, it is particularly troublesome in the design of high speed CMOS systems. With knowledge of the primary sources of noise, and the ways to combat that noise, it is possible to design and build systems which are electromagnetically quiet.

Very few references to specific values of capacitance, resistance, or inductance have been made in this document. The reason for this is that a value which works well in one application may not be effective in another. The best way to determine the values which will work well for a particular application is by experimentation.

Appendix A

COP8 DEVELOPMENT SYSTEM

iceMASTER™ COP8/400



TL/DD/12071-1

- **Full featured in-circuit emulator: MetaLink iceMASTER-400**
- **Low cost in-circuit emulator: MetaLink iceMASTER Debug Module**
- **Assembler/Linker/Librarian: National Semiconductor Assembler**
- **C Compiler: Bytecraft COP8C**
- **Fuzzy Logic Development Software: National Semiconductor NeuFuz**
- **Form Fit Function Emulators: National Semiconductor FFF emulators**
- **Emulator/OTP Programming Support: Certified PROM programmers**
- **Low Cost Evaluation and Programming Unit**

1 Introduction

1.1 MetaLink iceMASTER-400 Overview

The iceMASTER COP8/400 in-circuit emulator manufactured by MetaLink Corporation and marketed by National Semiconductor provides complete real-time emulation support for all members of the COP8 family. This stand-alone system is designed to provide maximum flexibility to the user through the interchangeable probe cards to support the various configurations and packages of the COP8 family. The interchangeable probe card connects to a common base unit which is linked with an IBM® PC® host through the RS-232 serial communications channel. Full assembly-level symbolic debugging is supported.

1.2 MetaLink iceMASTER-400

A detailed overview of the features and functions of the MetaLink iceMASTER software are provided in the following sections. Below is a list of available probecards and their ordering information. The minimum system configuration required for COP8 emulation consists of:

a. The MetaLink iceMASTER base unit with probe card or MetaLink iceMASTER debug module.

b. National Semiconductor's COP8 assembler package or Bytecraft COP8 C compiler.

c. IBM or compatible PC/AT®, 640k RAM, DOS 2.0 or higher.

NeuFuz™ and NeuFuz4™ are trademarks of National Semiconductor Corporation.
IBM®, PC® are registerd trademarks of International Business Machine Corporation.
Windows® is a registered trademark of Microsoft Corporation.
iceMASTER™ is a trademark of MetaLink Corporation.

MetaLink iceMASTER-400 Emulator Base Unit Ordering Information

Part Number	Description	Current Version
IM-COP8/400/1†	MetaLink base unit in-circuit emulator for all COP8 devices, symbolic debugger software and RS-232 serial interface cable, with 110V @ 60 Hz Power Supply.	Host Software; Ver. 3.3 Rev. 5, Model File Rev 3.050.
IM-COP8/400/2†	MetaLink base unit in-circuit emulator for all COP8 devices symbolic debugger software and RS-232 serial interface cable, with 220V @ 50 Hz Power Supply.	

†These parts include National's COP8 Assembler/Linker/Librarian Package (COP8-DEV-IBMA)

MetaLink iceMASTER-400 Probe Card Ordering Information (to be used with Base Unit)

Device	Package	Voltage Range	Probe Card
COP880C, 8780C	44 PLCC	4.5V–5.5V	MHW-880C44D5PC
		2.5V–6.0V	MHW-880C44DWPC
COP880C, 8780C	40 DIP	4.5V–5.5V	MHW-880C40D5PC
		2.5V–6.0V	MHW-880C40DWPC
COP881C, 8781C, 840C, 820C	28 DIP	4.5V–5.5V	MHW-880C28D5PC
		2.5V–6.0V	MHW-880C28DWPC
COP842C, 822C 8782C, 912C	20 DIP	4.5V–5.5V	MHW-880C20D5PC
		2.5V–6.0V	MHW-880C20DWPC
COP820CJ	28 DIP	4.5V–5.5V	MHW-820CJ28D5PC
		2.3V–6.0V	MHW-820CJ28DWPC
COP822CJ	20 DIP	4.5V–5.5V	MHW-820CJ20D5PC
		2.3V–6.0V	MHW-820CJ20DWPC
COP8640C, 8620C	28 DIP	4.5V–5.5V	MHW-8640C28D5PC
		2.5V–6.0V	MHW-8460C28DWPC
COP8642C, 8622C	20 DIP	4.5V–5.5V	MHW-8640C20D5PC
		2.5V–6.0V	MHW-8640C20DWPC
COP888CF	44 PLCC	4.5V–5.5V	MHW-888CF44D5PC
		2.5V–6.0V	MHW-888CF44DWPC
COP888CF	40 DIP	4.5V–5.5V	MHW-888CF40D5PC
		2.5V–6.0V	MHW-888CF40DWPC
COP884CF	28 DIP	4.5V–5.5V	MHW-884CF28D5PC
		2.5V–6.0V	MHW-884CF28DWPC
COP888CL	44 PLCC	4.5V–5.5V	MHW-888CL44D5PC
		2.5V–6.0V	MHW-888CL44DWPC
COP888CL	40 DIP	4.5V–5.5V	MHW-888CL40D5PC
		2.3V–6.0V	MHW-888CL40DWPC
COP884CL	28 DIP	4.5V–5.5V	MHW-884CL28D5PC
		2.3V–6.0V	MHW-884CL28DWPC

1 Introduction (Continued)

MetaLink iceMASTER-400 Probe Card Ordering Information (Continued)

Device	Package	Voltage Range	Probe Card
COP888CG, 888CS	44 PLCC	4.5V–5.5V	MHW-888CG44D5PC
		2.5V–6.0V	MHW-888CG44DWPC
COP888CG, 888CS, 888GG	40 DIP	4.5V–5.5V	MHW-888CF40D5PC
		2.5V–6.0V	MHW-888CG40DWPC
COP884CG, 884CS	28 DIP	4.5V–5.5V	MHW-884CG28D5PC
		2.5V–6.0V	MHW-884CG28DWPC
COP888EG	44 PLCC	4.5V–5.5V	MHW-888EG44D5PC
		2.5V–6.0V	MHW-888EG44DWPC
COP888EG	40 DIP	4.5V–5.5V	MHW-888EG40D5PC
		2.3V–6.0V	MHW-888EG40DWPC
COP884EG	28 DIP	4.5V–5.5V	MHW-884EG28D5PC
		2.3V–6.0V	MHW-884EG28DWPC
COP888GW	68 PLCC	2.5V–6.0V	MHW-888GW68PWPC
COP884BC	28 DIP	4.5V–5.5V	MHW-884BC28D5PC
COP888EK	44 PLCC	4.5V–5.5V	MHW-888EK44D5PC
		2.5V–6.0V	MHW-888EK44DWPC
	40 DIP	4.5V–5.5V	MHW-888EK40D5PC
		2.3V–6.0V	MHW-888EK40DWPC
COP884EK	28 DIP	4.5V–5.5V	MHW-884EK28D5PC
		2.3V–6.0V	MHW-884EK28DWPC

1.3 MetaLink iceMASTER Debug Module

The COP8 debug module is a low cost tool for designing, debugging, emulating and programming COP8 microcontrollers. Four versions of the Debug Module are available to support the most popular COP8 family members in their different pin configurations. The host for the Debug Module is a standard PC operating in a DOS environment, and is driven by the same menu driven user interface as the MetaLink iceMASTER-400. The Debug Module features an emulator with connectors available for connecting to various target boards. Additionally, the Debug Module offers the capability to program most of the EPROM versions of COP8 microcontrollers.

The minimum system configuration required for COP8 emulation consists of:

a. The MetaLink iceMASTER Debug Module

b. Target connector cables

c. External 5V and 13V power supplies

d. National Semiconductor's COP8 assembler package or Bytecraft COP8 C compiler

e. IBM or compatible PC, 640k RAM, DOS 2.0 or higher

1.4 MetaLink iceMASTER Evaluation and Programming Unit (EPU)

The COP8 EPU is an evaluation tool designed for simulating the basic family instruction set and for programming 40-DIP COP8780 or equivalent OTP/EPROM parts. In addition there is a target connector which reproduces the I/O of a COP880 device. The host for the EPU is a standard PC operating in a DOS environment and is driven by a simular user interface as the iceMASTER-400 and debug modules. The EPU offers an in circuit simulator with up to 4k program memory, 128 bytes of RAM.

1 Introduction (Continued)

MetaLink iceMASTER Debug Module Ordering Information (Low Cost Development Tool)

Part Number	Products Supported	Operating Voltage	Current Version
DM-COP8/880C†	COP880C/881C/882C, COP820/822C, COP840/842C, COP8780C/8781C/8782C	2.3V–6.0V	Host Software: Ver. 3.3 Rev. 5, Model File Rev 3.050.
DM-COP8/820CJ†	COP820CJ/822CJ	2.3V–6.0V	
DM-COP8/888CF†	COP888CF/884CF, COP888CL/884CL	2.3V–6.0V	Firmware: Ver. 6.07.
DM-COP8/888EG†	COP888CG/884CG, COP888CS/884CS, COP888EG/884EG	2.3V–6.0V	

†These parts include National's COP8 Assembler/Linker/Librarian Package (COP8-DEV-IBMA)

Target Connector Cables for Debug Module

Part Number	Description
DM-COP8/20D	20-Lead DIP Target Interface Cable
DM-COP8/28D	28-Lead DIP Target Interface Cable
DM-COP8/40D	40-Lead DIP Target Interface Cable
DM-COP8/44P	44-Lead PLCC Target Interface Cable

Adapter Kit for SO Packages for Use with Probe Cards and Debug Module Target Connection Cable

Part Number	Package
MHW-SOIC20	20 SO
MHW-SOIC28	28 SO

1.5 COP8 Assembler/Linker/Librarian Package

National Semiconductor offers a relocatable COP8 macro cross assembler. The assembler package includes a linker and librarian. It runs on industry standard compatible PCs in the DOS environment and generates full symbolic debugging information in the industry standard COFF (common object file format) format, which can be directly used in the MetaLink iceMASTER emulators. The Assembler package includes utilities to view the symbolic information embedded in the COFF file, and to generate a HEX or LM file for programming the COP8 parts.

The macro assembler generates relocatable object files with or without embedded symbolics. The librarian can be used to generate and maintain object libraries containing object files generated by the assembler. The linker is used to generate absolute download files from object files and object libraries.

Assembler/Linker/Librarian Ordering Information

Part Number	Description	Manual	Current Version
COP8-DEV-IBMA	COP8 Assembler/Linker/Librarian for IBM PC/AT or compatible	424421632-001	Ver. 4.4, Released September '94

1 Introduction (Continued)

1.6 COP8 C Compiler

National Semiconductor also offers the Bytecraft COP8C C-language compiler. This compiler features an expert system based optimizer and supports all members of the COP8 family of microcontrollers.

C-Compiler Ordering Information

Part Number	Description	Manual	Current Version
COP8C	Bytecraft COP8C compiler with Manuals for the IBM PC/AT or compatible	Bytecraft COP8C Manual	Ver. 1.0, Released Oct. '93

1.7 Fuzzy Logic Software

National Semiconductor offers NeuFuz, an automated development tool for generating fuzzy logic assembly code. NeuFuz runs on an IBM PC or equivalent on Microsoft Windows®. The input to NeuFuz is a text file containing system input-output data. NeuFuz learns the control surface represented by this data and automatically generates a fuzzy logic assembly language implementation. NeuFuz permits the user to simulate and fine tune the Fuzzy Logic control surface on the PC before it is plugged into the embedded processors.

Fuzzy Logic Software Ordering Information

Part Number	Description	Manual
NF2-C8A-KIT†	NeuFuz Learning Kit, with up to 2 analog inputs and 1 analog output. Generates COP8 assembly code.	42442645-001 or NF4-MAN
NF4-C8A†	NeuFuz4 software, with up to 4 analog inputs and 1 analog output. Generates COP8 assembly code.	42442645-001 or NF4-MAN
NF4-C8A-SYS†	NeuFuz4 software, with up to 4 analog inputs and 1 analog output. Generates COP8 assembly code. MetaLink COP8 Debug Module. It also includes Customer training and Application support.	42442645-001 or NF4-MAN

† These parts include National's COP8 Assembler/Linker/Librarian Package (COP8-DEV-IBMA)

1.8 Form Fit Function Emulator Support

National Semiconductor offers Form Fit Function (FFF) emulators for the basic and all the feature family members. The following table is the selection guide of emulators for the basic COP8 family members.

Form Fit Function Emulator Ordering Information (Basic Family)

Device Number	Package	Description	Emulates
COP8780CV	44 PLCC	One Time Programmable (OTP)	COP880C
COP8780CEL	44 LDCC	UV Erasable	COP880C
COP8780CN	40 DIP	OTP	COP880C
COP8780CJ	40 DIP	UV Erasable	COP880C
COP8781CN	28 DIP	OTP	COP881C, COP840C, COP820C
COP8781CJ	28 DIP	UV Erasable	COP881C, COP840C, COP820C
COP8781CWM	20 SO	OTP	COP881C, COP840C, COP820C
COP8782CN	20 DIP	OTP	COP842C, COP822C, COP912C
COP8782CJ	20 DIP	UV Erasable	COP842C, COP822C, COP912C
COP8782CWM	20 SO	OTP	COP8442C, COP822C, COP912C
COP8640CMHD COP8642CMHD	28 DIP 20 DIP	Hybrid, UV Erasable Hybrid, UV Erasable	COP8640C, COP8620C COP8642C, COP8622C
COP8720CJN COP8720CJWM COP8722CJWM	28 DIP 28 SO 20 SO	OTP OTP OTP	COP820CJ COP820CJ COP822CJ

Form Fit Function Emulator Ordering Information (Feature Family)

Device Number	Package	Description	Emulates
COP8788CLN-X COP8788CLN-R	40 DIP	OTP	COP888CL
COP888CLV-X COP888CLV-R	44 PLCC	OTP	COP888CL
COP884CLN-X COP884CLN-R	28 DIP	OTP	COP884CL
COP884CLWM-X COP884CLWM-R	28 SO	OTP	COP884CL
COP888CFN-X COP888CFN-R	40 DIP	OTP	COP888CF
COP888CFV-X COP888CFV-R	44 PLCC	OTP	COP888CF
COP884CFN-X COP884CFN-R	28 DIP	OTP	COP884CF
COP884CFWM-X COP884CFWM-R	28 SO	OTP	COP884CF
COP888EGN-X COP888EGN-R	40 DIP	OTP	COP888EG, 888CG, 888CS
COP888EGV-X COP888EGV-R	44 PLCC	OTP	COP888EG, 888CG, 888CS
COP884EGN-X COP884EGN-R	28 DIP	OTP	COP884EG, 884CG, 884CS
COP884EGWM-X COP884EGWM-R	28 SO	OTP	COP884EG,884CG, 884CS

Note: X = Crystal oscillator option
R = R/C oscillator option

1.9 COP8 Programming Support

The following programmers are certified for programming the One Time Programmable (OTP) and Form Fit Function (FFF) Emulator versions of COP8:

EPROM Programmer Information

Manufacter and Product	U. S. Phone Number	Europe Phone Number	Asia Phone Number
MetaLink Debug Module	(602) 926-0797	Germany: +49-8141-1030	Hong Kong: +852-737-1800
Xeltek-Superpro	(408) 745 7974	Germany: +49-2041-684758	Singapore: +65-276-6433
BP Microsystems- EP-1140	(800) 225-2102	Germany: +49-89-857-66-67	Hong Kong: +852-388-0629
Data I/O-Unisite; -System 29, -System 39	(800) 322-8246	Europe: +31-20-622866 Germany: +49-89-85-8020	Japan: +33-432-6991
Abcom- COP8 Programmer		Europe: +89-808707	
System General Turpro-1-FX; -APRO	(408)263-6667	Switzerland: +31-921-7844	Taiwan Taipei: +2-9173005

2.0 iceMASTER-400 Features

2.1 MetaLink iceMASTER Feature Overview

The MetaLink iceMASTER COP8 Model 400 In-Circuit Emulator for the COP8 family of microcontrollers features high-performance operation, ease of use, and an extremely flexible user-interface for maximum productivity. Interchangeable probe cards, which connect to the standard common base, support the various configurations and packages of the COP8 family.

The iceMASTER provides real time, full speed emulation up to 10 MHz, 32 kBytes of emulation memory and 4k frames of trace buffer memory The user may define as many as 32k trace and break triggers which can be enabled, disabled, set or cleared. They can be simple triggers based on code or address ranges or complex triggers based on code address, direct address, opcode value, opcode class or immediate operand. Complex breakpoints can be ANDed and ORed together. Trace information consists of address bus values, opcodes and user selectable probe clips status (external event lines). The trace buffer can be viewed as raw hex or as disassembled instructions. The probe clip bit values can be displayed in binary hex or digital waveform formats.

During single-step operation the dynamically annotated code feature displays the contents of all accessed (read and write) memory locations and registers, as well as flow-of-control direction change markers next to each instruction executed.

The iceMASTER's performance analyzer offers a resolution of better than 6 μs. The user can easily monitor the time spent executing specific portions of code and find "hot spots" or "dead code". Up to 15 independent memory areas based on code address or label ranges can be defined. Analysis results can be viewed in bargraph format or as actual frequency count.

Emulator memory operations for program memory include single line assembler, disassembler, view, change and write to file. Data memory operations include fill, move, compare, dump to file, examine and modify. The contents of any memory space can be directly viewed and modified from the corresponding window.

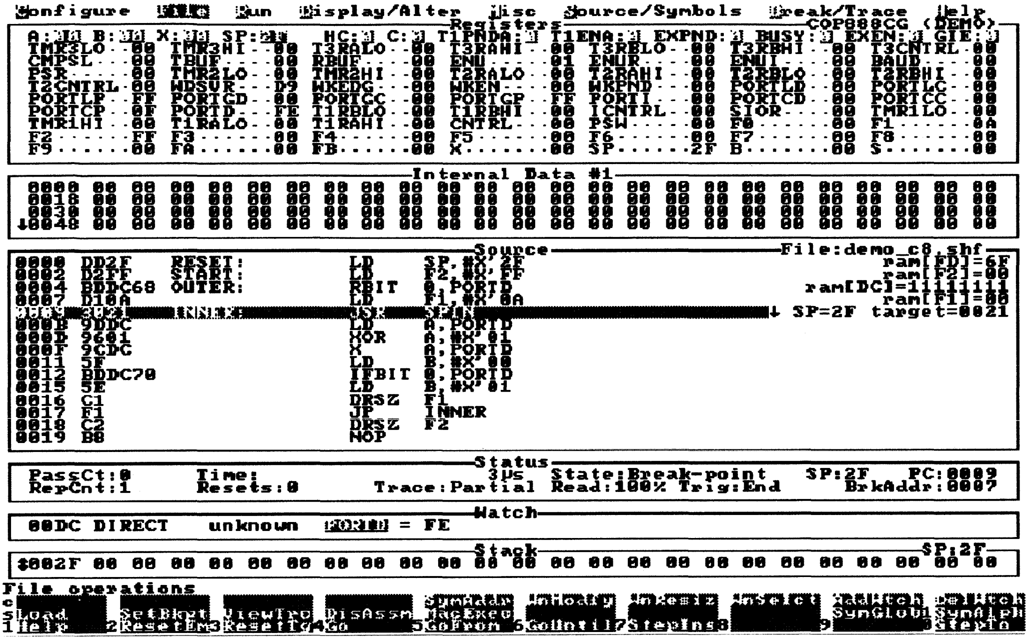
The iceMASTER comes with an easy to use windowed interface. Each window can be sized, highlighted, color-controlled, added, or removed completely Commands can be accessed via pull-down-menus and/or redefineable hot keys. A context sensitive hypertext/hyperlinked on-line help system explains clearly the options the user has from within any window.

2 iceMASTER-400 Features (Continued)

The iceMASTER connects easily to a PC via the standard COMM port and its 115.2 kBaud serial link keeps typical program download time to under 3 seconds.

For ordering information on the MetaLink iceMASTER system refer to the COP8 device datasheets or you may contact MetaLink directly under the following USA numbers:

Phone: (602) 926-0797
FAX: (602) 926-1198



```
                                   Registers                      COP888CG (DEMO)
 A:    B:    X:    SP:     HC:  C:  T1PNDA:  T1ENA:  EXPND:  BUSY:  EXEN:  GIE:
 TMR3LO··00 TMR3HI··00 T3RALO··00 T3RAHI··00 T3RBLO··00 T3RBHI··00 T3CNTRL·00
 CMPSL···00 TBUF····00 RBUF····00 ENU·····01 ENUR····00 ENUI····00 BAUD····00
 PSR·····00 TMR2LO··00 TMR2HI··00 T2RALO··00 T2RAHI··00 T2RBLO··00 T2RBHI··00
 T2CNTRL·00 WDSVR···D9 WKEDG···00 WKEN····00 WKPND···00 PORTLD··00 PORTLC··00
 PORTLP··FF PORTGD··00 PORTGC··00 PORTGP··FF PORTI···00 PORTCD··00 PORTCC··00
 PORTCP··0F PORTD···FE T1RBLO··00 T1RBHI··00 ICNTRL··00 SIOR····00 TMR1LO··00
 TMR1HI··00 T1RALO··00 T1RAHI··00 CNTRL···00 PSW·····00 F0······00 F1······0A
 F2······FF F3······00 F4······00 F5······00 F6······00 F7······00 F8······00
 F9······00 FA······00 FB······00 X·······00 SP······2F B·······00 S·······00

                              Internal Data #1
 0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
↓0048 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

                          Source                              File:demo_c8.shf
0000 DD2F    RESET:           LD     SP,#X'2F                      ram[FD]=6F
0002 D2FF    START:           LD     F2,#X'FF                      ram[F2]=00
0004 BDDC68  OUTER:           RBIT   0,PORTD                 ram[DC]=1111111
0007 D10A                     LD     F1,#X'0A                      ram[F1]=00
                                                          ↓ SP=2F  target=0021
000B 9DDC                     LD     A,PORTD
000D 9601                     XOR    A,#X'01
000F 9CDC                     X      A,PORTD
0011 5F                       LD     B,#X'00
0012 BDDC70                   IFBIT  0,PORTD
0015 5E                       LD     B,#X'01
0016 C1                       DRSZ   F1
0017 F1                       JP     INNER
0018 C2                       DRSZ   F2
0019 B8                       NOP

                              Status
 PassCt:0        Time:                3µs  State:Break-point   SP:2F    PC:0009
 RepCnt:1        Resets:0    Trace:Partial Read:100% Trig:End     BrkAddr:0007

                              Watch
 00DC DIRECT    unknown        = FE

                              Stack                                     SP:2F
 $002F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

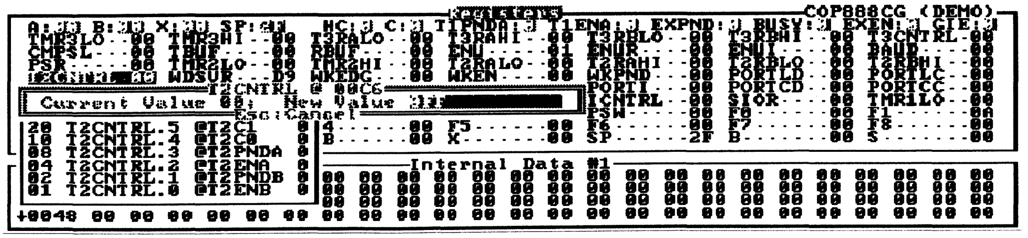
TL/DD/12071-2

Each window of the main screen can be sized, highlighted, color-controlled, positioned, added, or removed completely by using the "Configure | Window" menu. Commands can be accessed via pull-down-menus and/or redefineable hot keys. A context sensitive hypertext/hyperlinked on-line help system explains clearly the options the user has from within any window.

The main screen in this example has been configured to show the register, internal RAM data, source, status, watch and stack windows. A more detailed description of each of these windows will follow in this chapter.

Values can be changed directly within the window (with the exception of the source window).

A particular window can be selected by repeatedly pressing the TAB key or <shift>TAB keys.



```
                                   Registers                      COP888CG (DEMO)
 A:    B:    X:    SP:     HC:  C:  T1PNDA:  T1ENA:  EXPND:  BUSY:  EXEN:  GIE:
 TMR3LO··00 TMR3HI··00 T3RALO··00 T3RAHI··00 T3RBLO··00 T3RBHI··00 T3CNTRL·00
 CMPSL···00 TBUF····00 RBUF····00 ENU·····01 ENUR····00 ENUI····00 BAUD····00
 PSR·····00 TMR2LO··00 TMR2HI··00 T2RALO··00 T2RAHI··00 T2RBLO··00 T2RBHI··00
            WDSVR···D9 WKEDG···00 WKEN····00 WKPND···00 PORTLD··00 PORTLC··00
          T2CNTRL @ 00C6                     PORTI···00 PORTCD··00 PORTCC··00
 Current Value 00;   New Value               ICNTRL··00 SIOR····00 TMR1LO··00
              Esc:Cancel                     PSW·····00 F0······00 F1······0A
 20 T2CNTRL.5 @T2C1   0 4······00 F5······00 F6······00 F7······00 F8······00
 10 T2CNTRL.4 @T2C0   0 B······00 X·······00 SP······2F B·······00 S·······00
 08 T2CNTRL.3 @T2PNDA 0
 04 T2CNTRL.2 @T2ENA  0           Internal Data #1
 02 T2CNTRL.1 @T2PNDB 0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 01 T2CNTRL.0 @T2ENB  0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
+0048 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

TL/DD/12071-3

2 iceMASTER-400 Features (Continued)

The register window displays all of the dedicated registers of a specific—in this case the COP888CG—microcontroller. If the window is selected by pressing the TAB key, the user can select a particular register with the cursor key. If a register contains specific flags, these flags, their name and bit location are automatically displayed in another pop-up window, when this register is selected with the cursor keys.

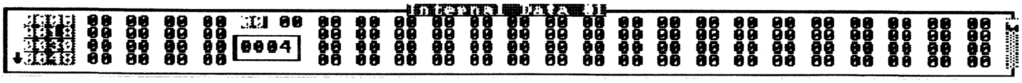
A register value can be changed by selecting the desired register with the cursor keys and then typing the new value in on the keyboard. Doing this will automatically pop up an edit window as soon as the first number key on the keyboard is pressed.

Note: Hex numbers starting with a letter must always be preceded by a "0" in order to distinguish those numbers from symbol names.

Each window can be scrolled once it is selected by using the cursor or page-up/page-down keys.

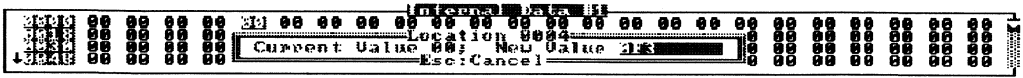
In the upper right corner of the window border the currently emulated chip is shown.

2.2.2 Internal RAM Data Window



```
Internal Data
0004
```

TL/DD/12071-4



```
Internal Data
Location 0004
Current Value 00;  New Value
Esc:Cancel
```

TL/DD/12071-5

Up to five different internal RAM windows can be displayed, each showing a different address range of RAM. In the window configuration the user can select between Hex only, ASCII only or Hex and ASCII combined display mode. The window may be scrolled when selected, so it is possible to scroll to any desired RAM location of the on-chip RAM.

When the RAM window is selected by pressing the TAB key, moving to a certain RAM location with the cursor keys automatically pops up a small window showing the current RAM address selected.

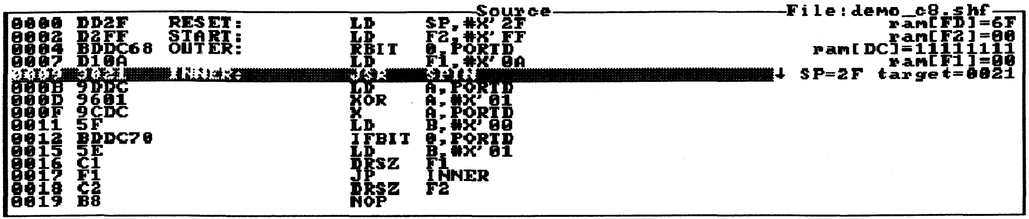
A RAM location can be changed by selecting the desired location with the cursor keys and then typing the new value in on the keyboard. Doing this will automatically pop up an edit window as soon as the first number key on the keyboard is pressed.

Note: Hex numbers starting with a letter must always be preceded by a "0" in order to distinguish those numbers from symbol names.

To modify a whole range of RAM the fill and modify functions of the "Display/Alter|RAM" menu can be used.

2 iceMASTER-400 Features (Continued)

2.2.3 Source Window



```
                                                Source                         File:demo_c8.shf
0000 DD2F    RESET:          LD     SP,#X'2F                                        ram[FD]=6F
0002 D2FF    START:          LD     F2,#X'FF                                        ram[F2]=00
0004 BDDC68  OUTER:          RBIT   0,PORTD                                   ram[DC]=11111111
0007 D10A                    LD     F1,#X'0A                                        ram[F1]=00
[illegible]                                                              ↓ SP=2F  target=0021
000B 9DDC                    LD     A,PORTD
000D 9601                    XOR    A,#X'01
000F 9CDC                    X      A,PORTD
0011 5F                      LD     B,#X'00
0012 BDDC70                  IFBIT  0,PORTD
0015 5E                      LD     B,#X'01
0016 C1                      DRSZ   F1
0017 F1                      JP     INNER
0018 C2                      DRSZ   F2
0019 B8                      NOP
```

TL/DD/12071-6

We have the option to display the source in a Higher Level Language, Disassembled Opcode/Operands, or a combination of both. The source window displays from left to right the following information:

Program memory address, COP8 Opcode, symbolic labels, disassembled Opcode (assembler mnemonics) and annotated code information (in the case of single-step or breakpoint).

A highlighted bar marks the location of the Program counter (i.e. the highlighted instruction is the next instruction to be executed, all previous displayed instructions have already been executed).

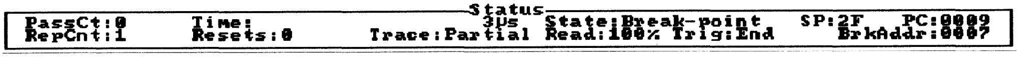
The annotated code displays the contents of all accessed (read and write) memory locations and registers, as well as flow-of-control direction change markers next to each instruction executed. The information shown refers to the memory location status BEFORE the instruction is executed. In the above shown screen the first instruction loads the Stackpointer with the value of 02F Hex. The Stackpointer is memory mapped at RAM address 0FD Hex. So the annotated code together with the instruction next to it shows that 02F was written to RAM address 0FD (the Stackpointer), which prior to that had the value of 06F.

The source window can be scrolled but no changes can be made within the source window. To change the program or to view a larger part of the disassembled code the "Display/Alter|Program memory" menu can be used.

In the upper right corner of the window border the filename of the currently loaded program file is displayed.

The source window permits the user to set and unset Breakpoints, switch between source modules, run until a particular label or address, change the source viewing mode (between Higher Level Language, Code and Mixed), search for a particular label or address, and assemble/disassemble code.

2.2.4 Status Window



```
                                          Status
 PassCt:0      Time:                        3µs  State:Break-point    SP:2F     PC:0009
 RepCnt:1      Resets:0           Trace:Partial  Read:100% Trig:End      BrkAddr:0007
```

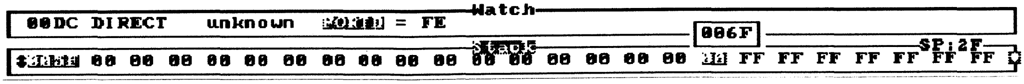
TL/DD/12071-7

The Status window shows information about the emulators current status, Pass counter, Repetition counter, number of resets, elapsed time, trace and breakpoint information and the current values of Stackpointer and Program counter.

The window is updated with every breakpoint, the elapsed time field is updated approximately every second and shows the total elapsed time since emulation has been started (after reset or a breakpoint). Hence when single stepping the time field displays the time used for the executed instruction. A hardware timer, located in the base unit of the emulator, is used for this and has an absolute error of 1 μs, so the times displayed for single instructions during single step might not always be accurate. When the timer is used for timing code sections, the accuracy of the timing is within ±1 μs.

The Pass counter, Repetition counter, Stackpointer and Program counter can be modified within the status window in the same way values in the RAM and Register windows are changed.

2.2.5 Watch/Stack Window



```
                                              Watch
 00DC DIRECT    unknown   [illegible] = FE
                                                                  006F
                                                                                  SP:2F
 $[illegible] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [illegible] FF FF FF FF FF FF FF
```

TL/DD/12071-8

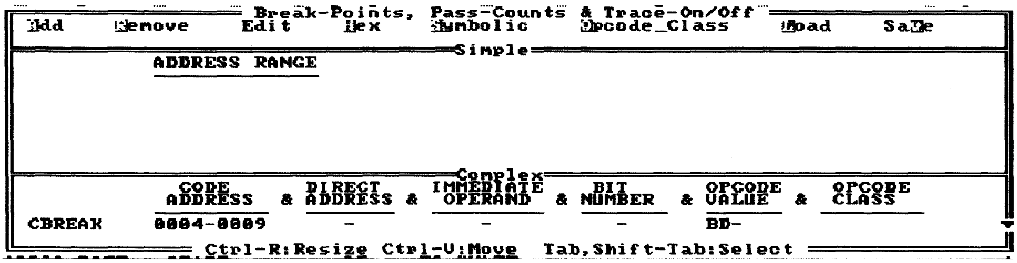
The Watch window allows the user to add memory locations of special interest for monitoring purposes during debugging. In the example shown, the D-Port has been added to the watch window and the information displayed shows:

The memory address, the symbol type ("direct" stands for a RAM/register address, whereas immediate would stand for an immediate value, which could also be added to the watch window, even though it would not make much sense), symbol name and symbol value (in the case of RAM/register the symbol value is the contents of RAM/register, in the case of an immediate the symbol value is the immediate itself).

Values in the Watch window can be modified in the same way values in the RAM and Register windows are changed.

The Stack window is simply a special RAM window with the actual stackpointer value shown in the upper right corner of the window border. It can be manipulated (scrolled, resized, moved, altered) in the same way as a RAM window.

2.3 Breakpoints/Trace Triggers



```
                 Break-Points, Pass-Counts & Trace-On/Off
Add     Remove     Edit     Hex     Symbolic     Opcode_Class     Load     Save
                                   Simple
           ADDRESS RANGE




                                   Complex
              CODE        DIRECT      IMMEDIATE      BIT      OPCODE     OPCODE
            ADDRESS   &  ADDRESS  &   OPERAND   &  NUMBER  &  VALUE   &  CLASS
CBREAK     0004-0009        -            -            -       BD-
             Ctrl-R:Resize Ctrl-U:Move   Tab,Shift-Tab:Select
```

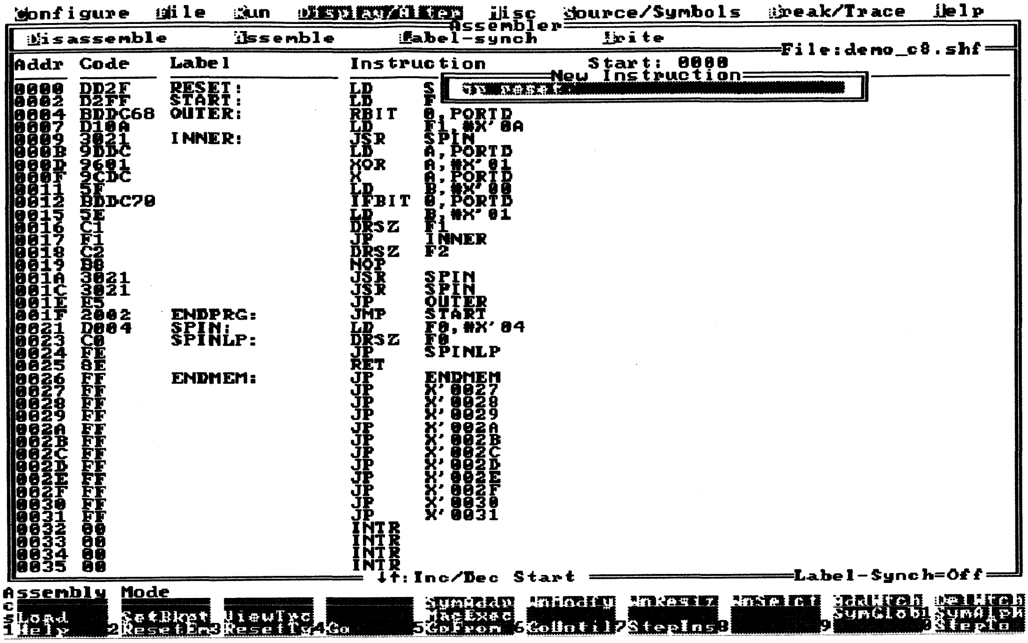
TL/DD/12071-9

Breakpoints and Trace triggers can be defined via the "Break/Trace|Set" menu, which pops up another window, and can range from simple triggers based on code address to very complex triggers. A simple breakpoint is set by selecting "Add|Cbreak" from the menu bar of the Breakpoint window.

A faster way of defining a simple breakpoint is by pressing the assigned hot key combination (Shift-F2 in the configuration shown in the example mainscreen figure).

Breakpoints can also be completly removed or temporarily disabled via the "Break/Trace menu".

2.4 Assembler/Disassembler



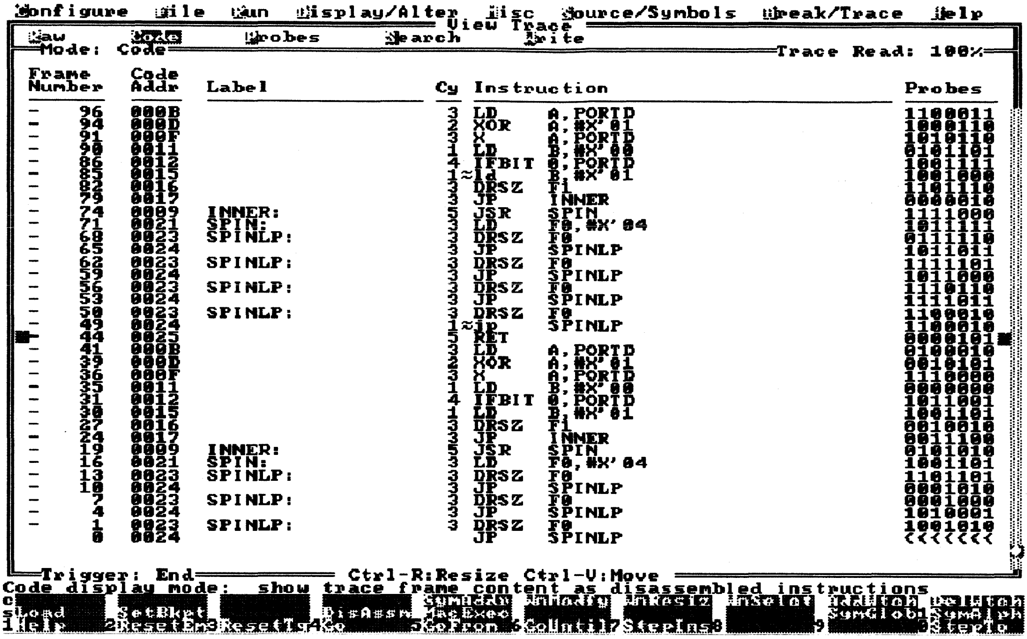
```
File:demo_c8.shf
Addr Code    Label        Instruction     Start: 0000
0000 DD2F    RESET:       LD     S
0002 D2FF    START:       LD     F
0004 BDDC68  OUTER:       RBIT   0,PORTD
0007 D10A                 LD     F1,#X'0A
0009 3021    INNER:       JSR    SPIN
000B 9DDC                 LD     A,PORTD
000D 9601                 XOR    A,#X'01
000F 9CDC                 X      A,PORTD
0011 5F                   LD     B,#X'00
0012 BDDC70               IFBIT  0,PORTD
0015 5E                   LD     B,#X'01
0016 C1                   DRSZ   F1
0017 F1                   JP     INNER
0018 C2                   DRSZ   F2
0019 B8                   NOP
001A 3021                 JSR    SPIN
001C 3021                 JSR    SPIN
001E E5                   JP     OUTER
001F 2002    ENDPRG:      JMP    START
0021 D004    SPIN:        LD     F0,#X'04
0023 C0      SPINLP:      DRSZ   F0
0024 FE                   JP     SPINLP
0025 8E                   RET
0026 FF      ENDMEM:      JP     ENDMEM
0027 FF                   JP     X'0027
0028 FF                   JP     X'0028
0029 FF                   JP     X'0029
002A FF                   JP     X'002A
002B FF                   JP     X'002B
002C FF                   JP     X'002C
002D FF                   JP     X'002D
002E FF                   JP     X'002E
002F FF                   JP     X'002F
0030 FF                   JP     X'0030
0031 FF                   JP     X'0031
0032 00                   INTR
0033 00                   INTR
0034 00                   INTR
0035 00                   INTR
```

TL/DD/12071-10

The iceMASTER's built in assembler/disassembler can be accessed via the Display/Alter menu. The user can view different areas of disassembled program code by specifying the start address or can make changes to the loaded code and save them to a file.

Note: Make sure that the "label-sync" feature is turned off when viewing program code, otherwise the disassembler might not disassemble correctly.

2.5 Trace View


Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
View Trace
Raw Code Probes Search Write
Ctrl-R:Resize Ctrl-V:Move
Load SetBkpt DisAssm MacExec SymGlob SymAlph
1Help 2ResetEm 3ResetTr 4Go 5GoFrom 6GoUntil 7StepIns 8 9 0

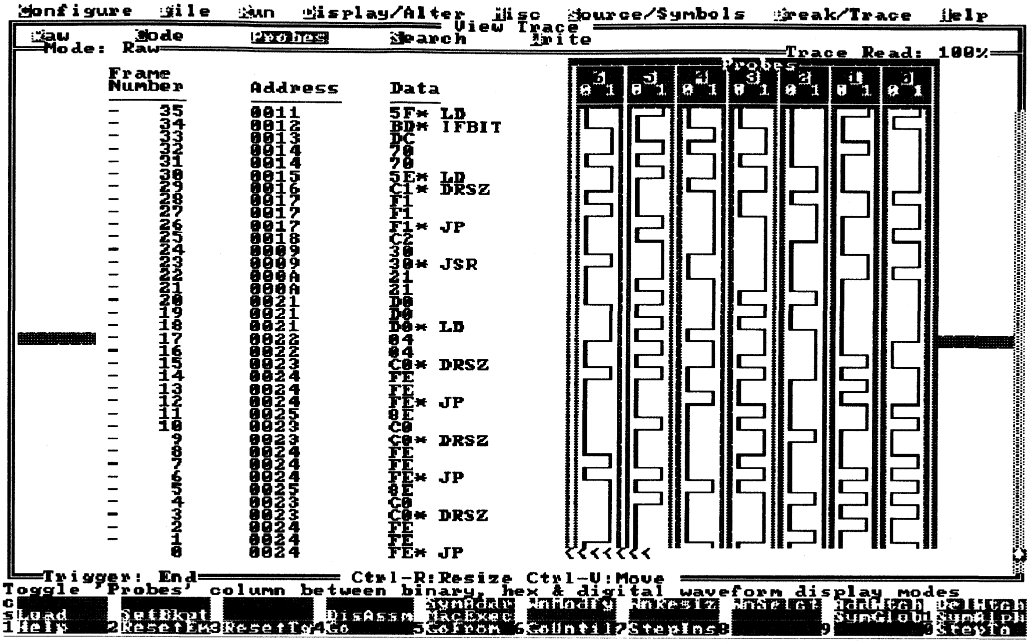
```
Mode: Code                                                    Trace Read: 100%

Frame    Code
Number   Addr   Label          Cy  Instruction               Probes

-   96   000B                  3   LD     A,PORTD            1100011
-   94   000D                  2   XOR    A,#X'01            1000110
-   91   000F                  3   X      A,PORTD            1010110
-   90   0011                  1   LD     B,#X'00            0101101
-   86   0012                  4   IFBIT  0,PORTD            1001111
-   85   0015                  1≈ld     B,#X'01            1001000
-   82   0016                  3   DRSZ   F1                 1101110
-   79   0017                  3   JP     INNER              0000010
-   74   0009   INNER:         5   JSR    SPIN               1111000
-   71   0021   SPIN:          3   LD     F0,#X'04           1011111
-   68   0023   SPINLP:        3   DRSZ   F0                 0111110
-   65   0024                  3   JP     SPINLP             1011011
-   62   0023   SPINLP:        3   DRSZ   F0                 1111101
-   59   0024                  3   JP     SPINLP             1011000
-   56   0023   SPINLP:        3   DRSZ   F0                 1110110
-   53   0024                  3   JP     SPINLP             1111011
-   50   0023   SPINLP:        3   DRSZ   F0                 1100010
-   49   0024                  1≈jp     SPINLP             1100010
-   44   0025                  5   RET                       0000101
-   41   000B                  3   LD     A,PORTD            0100010
-   39   000D                  2   XOR    A,#X'01            0010101
-   36   000F                  3   X      A,PORTD            1110000
-   35   0011                  1   LD     B,#X'00            0000000
-   31   0012                  4   IFBIT  0,PORTD            1011001
-   30   0015                  1   LD     B,#X'01            1001101
-   27   0016                  3   DRSZ   F1                 0010010
-   24   0017                  3   JP     INNER              0011100
-   19   0009   INNER:         5   JSR    SPIN               0101010
-   16   0021   SPIN:          3   LD     F0,#X'04           1001101
-   13   0023   SPINLP:        3   DRSZ   F0                 1101101
-   10   0024                  3   JP     SPINLP             0001010
-    7   0023   SPINLP:        3   DRSZ   F0                 0001000
-    4   0024                  3   JP     SPINLP             1010001
-    1   0023   SPINLP:        3   DRSZ   F0                 1001010
     0   0024                      JP     SPINLP             <<<<<<<

Trigger: End
Code display mode:  show trace frame content as disassembled instructions
```

TL/DD/12071-11

The Trace window occupies the whole screen and is brought up when the user selects Break/Trace|View from the menu bar. By default the trace buffer captures data in real-time whenever a program is run. It is organized as a circular buffer, so whenever the buffer is full, it will wrap around and start overwriting the oldest information.

The user can select among various display modes. Two examples are shown in this paragraph. Opcodes can be displayed in "Raw" or in "Code" format. Raw format lists each single instruction cycle as a separate entry, whereas Code format combines instruction cycles together to form one assembler mnemonic per entry line.

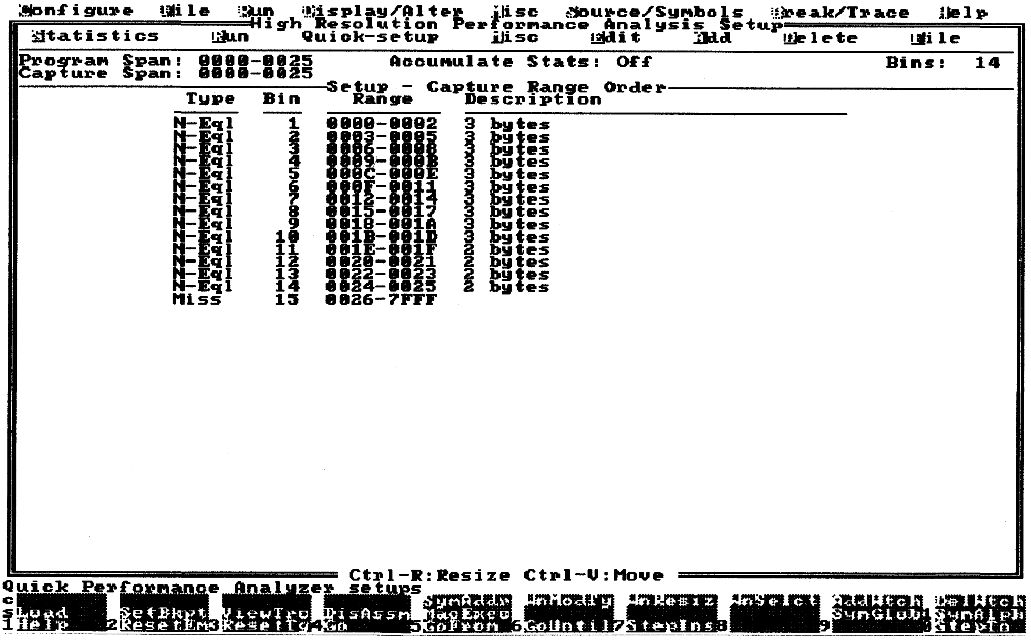
Apart from instruction cycles, address and opcode data, the Trace buffer also captures the status of eight external probe clips (external event lines). These are clips connected to the iceMASTER probe card, which can be connected to any type of TTL compatible signal. The display mode for the probe clips can be toggled between binary, hex and logic analyzer format and thus provide a nice little logic analyzer with a 1 μs resolution.


Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
View Trace
Raw Mode Probes Search Write
Probes
c
s Load SetBkpt DisAssm SymAddr MacExec MnModfy WnResiz WnSelct AddWtch SymGlobl DelWtch SymAlph
1 Help 2 ResetEm 3 ResetTr 4 Go 5 GoFrom 6 GoUntil 7 StepIns 8 9 0 StepTo

```
Mode: Raw                                              Trace Read: 100%
  Frame
  Number       Address      Data
  -  35        0011         5F*  LD
  -  34        0012         BD*  IFBIT
  -  33        0013         DC
  -  32        0014         70
  -  31        0014         70
  -  30        0015         5E*  LD
  -  29        0016         C1*  DRSZ
  -  28        0017         F1
  -  27        0017         F1
  -  26        0017         F1*  JP
  -  25        0018         C2
  -  24        0009         30
  -  23        0009         30*  JSR
  -  22        000A         21
  -  21        000A         21
  -  20        0021         D0
  -  19        0021         D0
  -  18        0021         D0*  LD
  -  17        0022         04
  -  16        0022         04
  -  15        0023         C0*  DRSZ
  -  14        0024         FE
  -  13        0024         FE
  -  12        0024         FE*  JP
  -  11        0025         8E
  -  10        0023         C0
  -   9        0023         C0*  DRSZ
  -   8        0024         FE
  -   7        0024         FE
  -   6        0024         FE*  JP
  -   5        0025         8E
  -   4        0023         C0
  -   3        0023         C0*  DRSZ
  -   2        0024         FE
  -   1        0024         FE
      0        0024         FE*  JP
Trigger: End          Ctrl-R:Resize Ctrl-U:Move
Toggle 'Probes' column between binary, hex & digital waveform display modes
```

TL/DD/12071-12

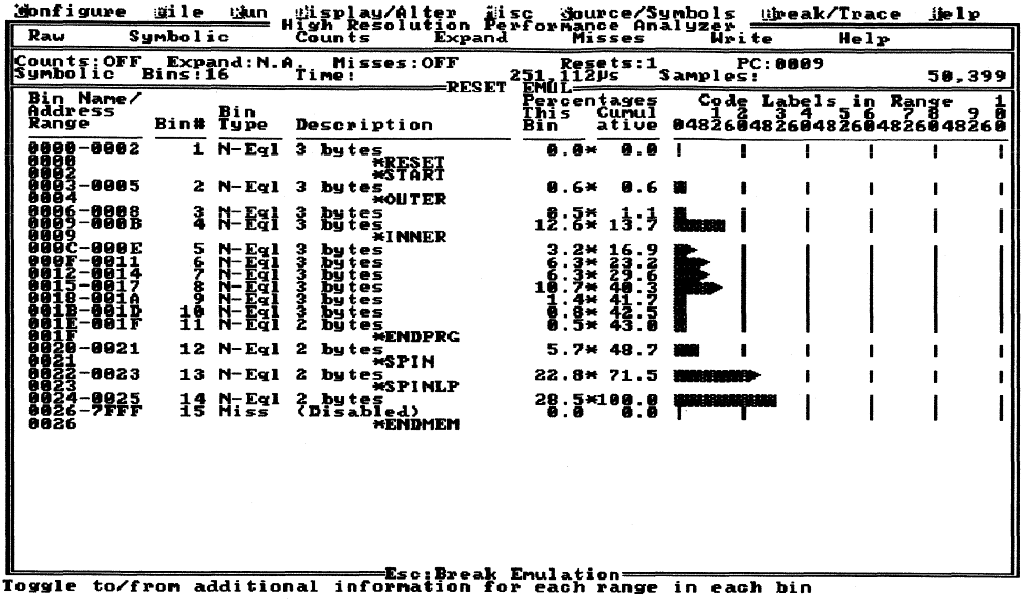
2.6 Performance Analyzer


Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
High Resolution Performance Analysis Setup
Statistics Run Quick-setup Misc Edit Add Delete File
c
s Load SetBkpt ViewTrc DisAssm SymAddr MacExec MnModfy WnResiz WnSelct AddWtch SymGlobl DelWtch SymAlph
1 Help 2 ResetEm 3 ResetTr 4 Go 5 GoFrom 6 GoUntil 7 StepIns 8 9 0 StepTo

```
Program Span: 0000-0025        Accumulate Stats: Off             Bins:  14
Capture Span: 0000-0025
                      Setup - Capture Range Order
            Type     Bin    Range        Description
            N-Eql      1    0000-0002    3 bytes
            N-Eql      2    0003-0005    3 bytes
            N-Eql      3    0006-0008    3 bytes
            N-Eql      4    0009-000B    3 bytes
            N-Eql      5    000C-000E    3 bytes
            N-Eql      6    000F-0011    3 bytes
            N-Eql      7    0012-0014    3 bytes
            N-Eql      8    0015-0017    3 bytes
            N-Eql      9    0018-001A    3 bytes
            N-Eql     10    001B-001D    3 bytes
            N-Eql     11    001E-001F    2 bytes
            N-Eql     12    0020-0021    2 bytes
            N-Eql     13    0022-0023    2 bytes
            N-Eql     14    0024-0025    2 bytes
            Miss      15    0026-7FFF
                      Ctrl-R:Resize Ctrl-V:Move
Quick Performance Analyzer setups
```

TL/DD/12071-13

The iceMASTER's performance analyzer is accessed via the Misc menu and provides also a "quick setup" feature, which distributes the sample bins automatically over the entire program range, as shown in above example. Apart from that the user can monitor specific ranges of the program by distributing the sample points manually.

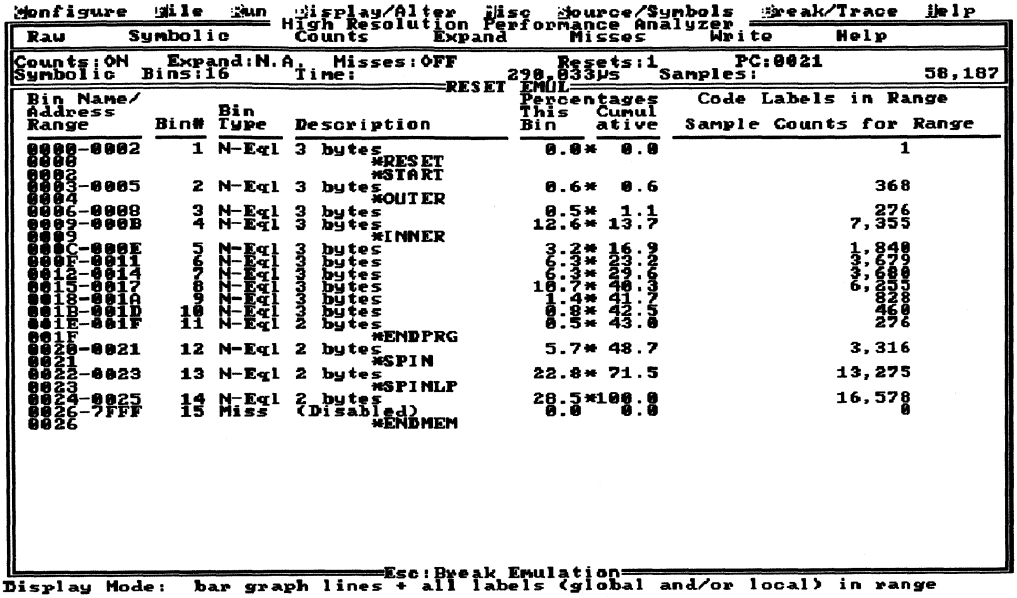


```
Configure  File  Run  Display/Alter  Misc  Source/Symbols  Break/Trace  Help
                     High Resolution Performance Analyzer
 Raw      Symbolic      Counts      Expand      Misses      Write      Help

Counts:OFF  Expand:N.A.  Misses:OFF          Resets:1           PC:0009
Symbolic   Bins:16       Time:           251,112µs   Samples:            50,399
                                        RESET EMUL
 Bin Name/                                   Percentages    Code Labels in Range   1
 Address           Bin                       This   Cumul     1 2  3 4  5 6  7 8  9 0
 Range       Bin#  Type   Description        Bin    ative  04826048260482604826048260
 0000-0002     1   N-Eql  3 bytes             0.0*   0.0
 0000                         *RESET
 0002                         *START
 0003-0005     2   N-Eql  3 bytes             0.6*   0.6
 0004                         *OUTER
 0006-0008     3   N-Eql  3 bytes             0.5*   1.1
 0009-000B     4   N-Eql  3 bytes            12.6*  13.7
 0009                         *INNER
 000C-000E     5   N-Eql  3 bytes             3.2*  16.9
 000F-0011     6   N-Eql  3 bytes             6.3*  23.2
 0012-0014     7   N-Eql  3 bytes             6.3*  29.6
 0015-0017     8   N-Eql  3 bytes            10.7*  40.3
 0018-001A     9   N-Eql  3 bytes             1.4*  41.7
 001B-001D    10   N-Eql  3 bytes             0.8*  42.5
 001E-001F    11   N-Eql  2 bytes             0.5*  43.0
 001F                         *ENDPRG
 0020-0021    12   N-Eql  2 bytes             5.7*  48.7
 0021                         *SPIN
 0022-0023    13   N-Eql  2 bytes            22.8*  71.5
 0023                         *SPINLP
 0024-0025    14   N-Eql  2 bytes            28.5* 100.0
 0026-7FFF    15   Miss   (Disabled)          0.0    0.0
 0026                         *ENDMEM

                          Esc:Break Emulation
Toggle to/from additional information for each range in each bin
```

TL/DD/12071–14

The performance analyzer's display mode can be switched between a bargraph display or a display of actual counts. The user can select to include symbol label information for easier monitoring.

2 iceMASTER-400 Features (Continued)



```
Configure  File  Run  Display/Alter  Misc  Source/Symbols  Break/Trace  Help
                      High Resolution Performance Analyzer
 Raw      Symbolic      Counts      Expand      Misses      Write      Help

Counts:ON    Expand:N.A.   Misses:OFF           Resets:1        PC:0021
Symbolic  Bins:16        Time:          290,033µs  Samples:                58,187
                                        RESET EMUL
 Bin Name/                                       Percentages   Code Labels in Range
 Address            Bin                          This   Cumul
 Range        Bin#  Type   Description           Bin    ative  Sample Counts for Range

 0000-0002      1  N-Eql  3 bytes                  0.0*   0.0                    1
 0000                              *RESET
 0002                              *START
 0003-0005      2  N-Eql  3 bytes                  0.6*   0.6                  368
 0004                              *OUTER
 0006-0008      3  N-Eql  3 bytes                  0.5*   1.1                  276
 0009-000B      4  N-Eql  3 bytes                 12.6*  13.7                7,355
 0009                              *INNER
 000C-000E      5  N-Eql  3 bytes                  3.2*  16.9                1,840
 000F-0011      6  N-Eql  3 bytes                  6.3*  23.2                3,679
 0012-0014      7  N-Eql  3 bytes                  6.3*  29.6                3,680
 0015-0017      8  N-Eql  3 bytes                 10.7*  40.3                6,255
 0018-001A      9  N-Eql  3 bytes                  1.4*  41.7                  828
 001B-001D     10  N-Eql  3 bytes                  0.8*  42.5                  460
 001E-001F     11  N-Eql  2 bytes                  0.5*  43.0                  276
 001F                              *ENDPRG
 0020-0021     12  N-Eql  2 bytes                  5.7*  48.7                3,316
 0021                              *SPIN
 0022-0023     13  N-Eql  2 bytes                 22.8*  71.5               13,275
 0023                              *SPINLP
 0024-0025     14  N-Eql  2 bytes                 28.5*100.0               16,578
 0026-7FFF     15  Miss   (Disabled)               0.0    0.0                    0
 0026                              *ENDMEM

                              Esc:Break Emulation
Display Mode:  bar graph lines + all labels (global and/or local) in range
```

TL/DD/12071-15

3 iceMASTER Software Installation

3.1 Installation Procedures

The MetaLink iceMASTER system comes with two disks of software, one contains the host software for the iceMASTER base unit and the other the probe card software. The software is installed by inserting the disk label "insert 1st" into drive D: and typing:

```
D: install D: c:\path
```

The installation process is menu driven and should therefore be self-explanatory.

The same procedure is repeated for the second disk.

When installing the probe card software the user has to specify whether it is an iceMASTER-400 or Debug Module, which chip is to be emulated and the installation program then creates automatically the appropriate help, help index and model files for that chip. If a debug module is specified, then the PROM programmer menu items are activated in the iceMASTER software.

3.2 Emulating Different COP8 Family Members

The model file created by the installation program defines chip specific features, such as ROM size, RAM size, special function registers, etc. for the user specified chip. If the user wants to emulate a different COP8 family member the model and help files for this chip have to be created. Rather than doing a new installation of the software the MF__GEN utility program provided with the iceMASTER software can be used. Executing that program will present a menu to the user where the desired COP8 device can be selected.

4 Creating an Emulator Download File

4.1 Emulator Download File Formats

To make full use of the iceMASTER's symbolic debug capabilities the user has to create a Common Object File Format (COFF), or Binary Code File (COD), or Symbolic Hex File (SHF). These files have symbolic debug information embedded in them. The steps that have to be performed to generate the COFF/COD/SHF are outlined in this chapter.

4.2 Generating A Common Object File Format File (COFF)

4.2.1 Running the Assembler/Linker (Ver. 4.0 or later)

The MetaLink system uses the National Semiconductor's COP8 assembler/linker/librarian (Version 4.0 or later) to generate the COFF file. The procedure to create the COFF file with the symbolic information and source lines embedded in the COFF file is accomplished by runnning the assembler and linker with the following parameters:

```
ASMCOP filename /SYMBOLDEBUG /COMM
/LOCALSYMBOLS /TABLESYMBOLS /L
```

The default filename extension is .ASM and does not need to be specified. The assembler generates a relocatable object file with extension .OBJ. The option /SYMBOLDEBUG includes symbolic debugging information in the object file. The option /COMM includes source lines as comments in the object file. The options /LOCALSYMBOLS and /TABLESYMBOLS includes the symbol table in the object file with all the local symbols also.

```
LNCOP filename /FORMAT = COF /T
```

The default filename extension for the linker is .OBJ and does not need to be specified.The linker generates a COFF output file with default extension .COF when used with the above options. The option /T includes the symbol table.

4.3 Generating A Binary Code File (COD) File

The MetaLink system also uses the Bytecraft COP8C compiler to generate the COD file, which contains embedded symbolic debugging information.

4.3.1 Running the Bytecraft COP8C Compiler

A C application program written for COP8C compiler can be compiled using the COP8C compiler using the following command line options:

```
COP8C filename +da +l +x
```

The default filename extension is .C and does not need to be specified. The option +da produces an ASCII format hex dump file, +l generates a listing file and +x generates the cross reference file. By default a code file is generated with name "filename.cod". The COD file generated can be directly downloaded into the MetaLink iceMASTER or Debug Module.

4.4 Generating A Symbolic Hex File (SHF)

4.4.1 Running the Absolute Assembler (Rev. E or earlier)

The MetaLink system uses National Semiconductor's COP8 assembler (Rev. E) to create a symbol file, listing file and object code file. However, the current version of the assembler does not distinguish between RAM/register symbols and immediate/bit symbols, but distinguishes only between label symbols and all other type of symbols. Therefore the user has to supply the missing information to the MetaLink system by means of an exception file which is handled in more detail later on in this chapter.

To create the necessary output files the assembler is called with the following parameters:

```
ASM800 filename /l=filename.lst /S
```

The default filename extension is .MAC and doesn't need to be specified.

The /l switch creates a listing file with the file extension .LST, the /s switch creates a symbol file. The object file is created automatically with the extension .LM (NSC Hex format).

In the next step the .LM object file has to be converted into an Intel Hex file for use with the MetaLink system. This is done with the LMHEX utility provided with the assembler:

```
LMHEX filename
```

This will create an Intel Hex file with the extension .HEX

4.4.2 Creating an Exception File

Once the program has been successfully assembled, the user has to create an exception file. The exception file contains all symbols that are not address labels and has the following format:

symbolnameidentifier

where identifier is either 2, 5 or 9, with:

2 = RAM/register address,
5 = immediate or bit value,
9 = don't care.

Example exception file:

```
PORTD 2
PORTGD 2
GIE 5
JUNK 9
```

Note: To get the exception file definition for all COP8 standard register and bit symbol names the user should copy the DEMO__C8.EXC file which comes with the iceMASTER software to a file called filename.exc of the assembled program and then edit and add the missing new symbol names to that file.

Once this exception file is created, a Symbolic Hex File can be created.

4.4.3 Creating a Symbolic HEX File

A Symbolic Hex File (SHF) is created with the MetaLink utility program MKSHF, which takes the listing, hex and exception files as input and creates a symbolic hex file (.SHF extension) as output. The syntax is:

```
MKSHF filename.lst filename.hex
filename.exc filename.shf
```

The symbolic hex file is downloaded to the MetaLink system and contains all necessary program and symbol information.

4.4.4 Using a Batch File

A Batch file can be used to perform most of the steps (apart from creating the exception file) described in this chapter automatically This batch file can also be used to do the initial assembly of the program as the LMHEX and MK_SHF utility programs will simply fail if the necessary input files are not present. Following is the listing of the MAKE.BAT file:

```
echo off
rem syntax: make filename [path]
rem specification of path is optional, no filename-extension
rem is entered on the command line
rem change path
cd %2
rem assemble file (default file-extension = .MAC)
rem if you want to use a different extension, you have to
rem insert it after the "asm800 %l", e.g "asm800 %l.ASM"
rem create printfile: FILENAME .LST and symbol file
rem FILENAME: SYM
asm800 %l /l= %l.lst /s
rem convert .LM file to Intel Hex format .HEX
lmhex %l
rem make symbolic hex file
rem input files are:
rem FILENAME.SYM ;symbol file
rem FILENAME.EXC ;exception file, see readme files for
rem how to create an exeption file
rem FILENAME.HEX ;object file in Intel Hex format
rem output file is:
rem FILENAME.SHF ;symbolic hex file which is downloaded to
rem MetaLink system
mkshf %l.sym %l.exc %l.hex %l.shf
echo on
```

5 MetaLink iceMASTER-COP8 Debug Module

5.1 MetaLink iceMASTER-COP8 Debug Module Overview

The MetaLink iceMASTER-COP8 Debug Module has most of the salient features of the iceMASTER-400 and in addition has a COP8 PROM programmer. It is a tool for designing, debugging and evaluating COP8 Microcontroller Unit (MCU) devices. This provides all of the essential MCU timing, I/O circuitry and therefore simplifies the evaluation of the prototype hardware/software product.

The Debug Module is controlled by an IBM PC (or compatible) running MS-DOS communicating over a serial port at 9600 baud. The Debug Module uses the same menu driven user interface as the MetaLink iceMASTER-400 (Refer to Section 2 for more details).

The Debug Module can be connected to a target system in place of the microcontroller (using an optional target interface cable) or operated independently in the stand alone mode. Stand alone mode allows you to emulate hardware and/or execute code without a target system (provided no interaction with external devices is needed).

Hardware designers can use the Debug Module to develop and debug their designs. All available features of a given device are accessible interactively, as well as through the application programs. Software designers have complete emulation capability as well. The Debug Module will execute the code just like the real part because it uses a real part for emulation.

5.2 Emulation Characteristics Of The Debug Module

The Debug Module utilizes the INTR instruction to implement software breakpoints. If your application code contains any INTR instructions, they will never be executed. The debug module behaves exactly as though a breakpoint occurred when it passes through such locations. In the iceMASTER-400 breakpoints are implemented in hardware and if the application were to contain INTR instructions, they will not be treated as breakpoints.

When a breakpoint is set on an instruction which could be potentially skipped, the emulation will break only when the instruction is actually executed. Emulation will never break when the instruction is skipped. When a breakpoint is set on an instruction which could be potentially skipped and that instruction is skipped during emulation, the address and data values captured in the trace will be different from a real COP8 processor. The first cycle will be that of a skipped INTR instruction. For multi-byte instructions, subsequent cycles will be the same as that of executed NOP instructions.

When a breakpoint is set on an instruction, emulation breaks **before** the instruction executes. The first two bytes of the stack will be overwritten when the breakpoint is reached. This is because INTR instructions are used to implement breakpoints.

For the COP880 family of processors, the timers are shut off at breakpoints shortly after emulation stops. When the emulation resumes, the timers are restarted just before the emulation of the target application program actually begins. In the COP884/COP888 family there is no delay in the stopping or restarting running timers upon reaching a breakpoint or when emulation resumes.

To allow for clock resynchronization in the COP microcontroller, it is necessary to program two NOP instructions immediately after the processor comes out of the HALT mode. When the multi-input wakeup interrupt is enabled, the first two instructions of the interrupt routine must be NOP's. If no interrupts are used to enter the HALT mode, then two NOP's must follow "enter HALT mode" (set G7 data bit) instruction.

As with the HALT mode, it is necessary to program two NOP's to allow clock resynchronization upon return from the IDLE mode. The NOP's are placed either at the beginning of the IDLE timer interrupt routine or immediately following the "enter IDLE" mode instruction.

5.3 Prom Programmer

The COP8 Debug Module can program the code memory in most of the supported EPROM or HYBRID devices. Operation of the programmer is under control of the host software from the *Misc|PROM Programmer* pull down menu. To run the EPROM programmer a EPROM programming voltage V_{PP} (nominally 13V) must be supplied. There is an option to use an external V_{PP} source or use the on-board V_{PP} generator. The on-board V_{PP} generator is built by populating the board with the components marked on the Debug Module and adjusting the potentiometer to 13.0V.

All programming operations use the code loaded into the Debug Module from the *File|Load* command. The programming buffer shares the same buffer space as the emulation memory. Therefore it is possible to load code into the buffer in COF and COD formats in addition to the standard HEX formats. It is also possible to read code from the device in the programming socket. Code read from the device will be written into the emulation memory in the Debug Module, and may not match the labels from a previously loaded program.

By default, areas in the emulation memory with no code will be filled with 0x00 hex bytes; however, you can use *Configure|Options|(Miscellaneous) Code Init* command to change the 0x00 hex byte to any other byte value.

5 MetaLink iceMASTER-COP8 Debug Module (Continued)

The *Misc |PROM |Programmer |EPROM |Program* and *Misc |PROM |Programmer |EPROM Auto* functions verify each byte as it is programmed, and the entire area at completion, so it is not necessary to separately *Verify* the parts after programming.

To accomplish correct programming of the EPROM'd COP8 devices the following procedure must be followed:

1. Select the correct device type from the *Misc |PROM Programmer |Device* selection.
2. Ensure that the device is blank. To accomplish this, subject the windowed COP8 device to a UV light source for the stipulated amount of time. Then perform the blank check from *Misc |PROM Programmer |EPROM |Blank Check* command.
3. Load the correct application program into the emulation buffer. This can be done by loading the application from the *Misc |PROM Programmer | Code |Load* command.
4. If the device has an ECON register (not present in the HYBRID devices), then select the clock option, security option and the RAM size option from the *Misc |PROM Programmer |Configuration* menu.
5. Program the device either by running *Misc |PROM Programmer |Auto* or *Misc |PROM Programmer | EPROM |Program* command.
6. If the device has an ECON register then run *Misc |PROM Programmer |Register |Program*. This ensures that the ECON register is correctly programmed.

Note: In HYBRID parts, the Options register contents have to be embedded into the assembly program before the part is programmed. There is no menu selection for the contents of the Options register in the Debug Module.

6 MetaLink Evaluation and Programming Unit

6.1 EPU Overview

The iceMASTER-COP8 Evaluation and Programming Unit is a low cost, In-Circuit Simulator which can be used to debug code and hardware designs for the COP880C microcontroller and to program the COP8780 and COP87M80 EPROM/OTP microcontrollers. The host computer for the EPU-COP8 is a standard PC (or compatible) running the DOS Operating System. The interface to the EPU-COP8 is over the RS-232C serial channel at 115,200 baud.

COP8 code can be generated using the COP8 assembler/linker provided, or the very efficient C Compiler, COP8C, available from Byte Craft Limited. Once your code is loaded you can quickly move through the software and hardware evaluation process, while providing complete control over the microcontroller. The EPU-COP8 is a cost efficient system that offers an easy introduction to the COP8 family of microcontrollers. The EPU-COP8 offers the same user interface as the MetaLink iceMASTER-COP8 Debug Module and the MetaLink iceMASTER-COP8/400 emulator, allowing for an easy migration to the low cost Debug Module or to a full featured ICE.

6.2 EPU PROM Programmer

After completing the debugging process, you may program your code into the COP8780 or COP87M80 EPROM/OTP microcontroller using the EPU-COP8. The on-board voltage generator supplies all the voltages required to program the EPROM/OTP using only the wall mounted power supply provided. This feature provides an easy transition from In-Circuit Simulation to real-time target test mode.

7 Using NeuFuz4

7.1 INSTALLATION PROCEDURES

The NeuFuz4 software can be installed from the distribution disk into drive A by typing:

```
A:install c:\n4dir
```

Where "c:\n4dir" is the directory where the executables are copied into. The installation process is menu driven and should therefore be self-explanatory.

After this invoke MS-Windows and create a new Program Item from the Program Manager and enter the following at the Command:

```
c:\n4dir\n4.exe
```

At the description enter:

```
NeuFuz4
```

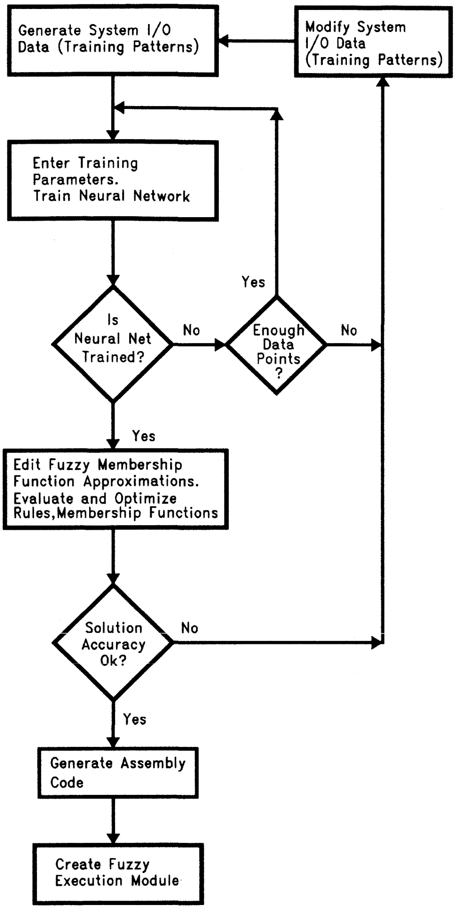
Then click on OK and the Program Manager will create an icon called NeuFuz. To run NeuFuz click twice on the NeuFuz4 icon.

7.2 GENERAL DESCRIPTION

NeuFuz4 is a software design system, based on National Semiconductor's proprietary NeuFuz technology. It learns a system behavior and then automatically generates Fuzzy Logic Rules and Membership Functions. Determining a proper set of Fuzzy Rules and Membership Functions needed to adequately describe a system behavior is the most difficult step in Fuzzy Logic design. NeuFuz4 simplifies this task by using Neural Networks learning and generalization capability to accomplish this task. NeuFuz4 also provides graphical on-line capabilities to verify, tune and optimize the Fuzzy Logic design model. NeuFuz4 can also be used as a general purpose learning system with the solution implemented in Fuzzy Logic. Additionally, NeuFuz4 translates Fuzzy Logic designs into COP8 Assembly code automatically. This code may then be assembled for a variety of target COP8 processors.

7.3 THE NeuFuz DEVELOPMENT PROCESS

NeuFuz4 uses a modified Back Propagation algorithm to train the multilayered feedforward Neural Network. A set of input-output data (that covers the entire range of system operation) along with the user selected training parameters are fed into the Neural Network. The Neural Network trains iteratively by back propagation thereby altering the weights for the interconnection between neurons (This is also known as learning.). When the training process is completed, the output of the Neural Network is translated into Fuzzy Logic Rules and bell shaped Membership Functions. NeuFuz4's Neural Network is specially constructed to map its knowledge directly into Fuzzy Logic. This enables NeuFuz to automatically generate Fuzzy Rules and Membership Functions that describe the trained system. The user has the choice to do approximations and implement the Fuzzy solutions on an inexpensive, low performance embedded controller, thereby sacrificing some accuracy or implement the bell shaped Membership Functions directly on a high performance embedded controller, generating a more accurate result. NeuFuz4 has the ability to approximate Membership Functions with traditional shapes (such as triangles, trapezoids and polygons).


Generate System I/O Data (Training Patterns)
Modify System I/O Data (Training Patterns)
Enter Training Parameters. Train Neural Network
Is Neural Net Trained?
No
Enough Data Points ?
Yes
No
Yes
Edit Fuzzy Membership Function Approximations. Evaluate and Optimize Rules,Membership Functions
Solution Accuracy Ok?
No
Yes
Generate Assembly Code
Create Fuzzy Execution Module

TL/DD/12071-16

NeuFuz Based Development Process

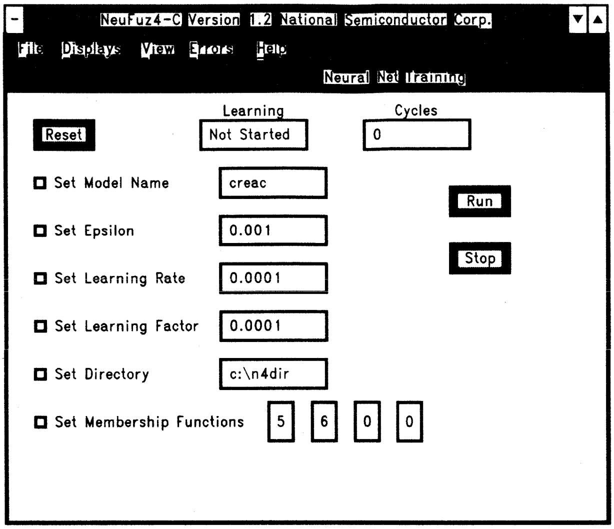
A highly intuitive graphical user interface enables the user to verify and optimize the Fuzzy solution. The user can eliminate those Rules which contribute less to the solution while having total control over the accuracy of the solution.

The final step of the design process is porting the Fuzzy solution to an embedded processor. NeuFuz4 supports this via the automatic code generator. It generates COP8 Assembly code that may be compiled for various COP8 processors.

1. Neural Network Training.
2. Fuzzy Rules and Membership Functions Generation.
3. Fuzzy Rule Verification and Optimization.
4. Assembly Code Generation.
5. Fuzzy Execution Module Creation.

7.4 Neural Network Training

The first step in using NeuFuz4 is to create an input ASCII data file containing the system training patterns, where each line is treated as a pattern. Each pattern is in the form of the inputs to the system followed by the output. The training pattern can have up to four input values and one desired output value.


NeuFuz4-C Version 1.2 National Semiconductor Corp.
File Displays View Errors Help
Neural Net Training
Reset
Run
Stop

Learning: Not Started
Cycles: 0

□ Set Model Name: creac

□ Set Epsilon: 0.001

□ Set Learning Rate: 0.0001

□ Set Learning Factor: 0.0001

□ Set Directory: c:\n4dir

□ Set Membership Functions: 5 6 0 0

TL/DD/12071–17

The Neural Network Training window is used to control the training process. The training parameters for the Neural Network training are specified within this window: Membership Functions, Epsilon, Learning Rate and Learning Factor. The structure and the training behavior of the Neural Network are determined by these parameters. Below is a brief description of the Neural Network training parameters.

7.4.1 Membership Functions

This is the number of Membership Functions used to divide the input space. Each input can be represented from 2 up to 7 Membership Functions. Selection of more Membership Functions will generate a solution with higher resolution and greater accuracy.

7.4.2 Learning Rate and Learning Factor

These parameters govern the Neural Network training. Setting and changing these parameters appropriately is a key to quick and accurate training of the Neural Network.

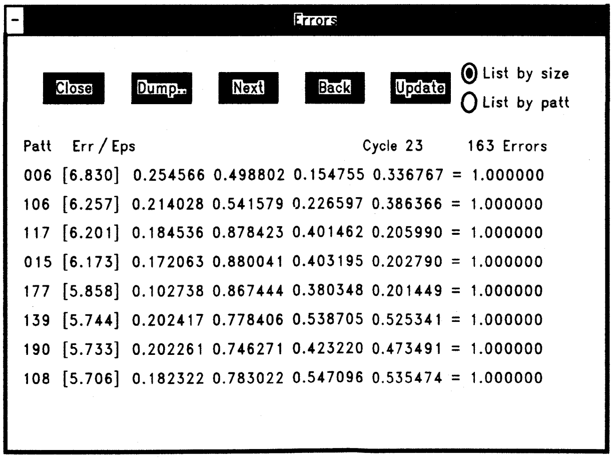
7.4.3 Epsilon

This is the absolute maximum error allowed in the Neural Network on completion of training. This parameter affects the rate of convergence of the Neural Network training and accuracy of the solution.

For useful instructions on how to select the appropriate epsilon, learning rate and learning factor, see the NeuFuz user's manual.

7.4.4 Error Window

When the Neural Network is being trained to learn the system behavior, it is important to monitor the error window. This window will provide very timely and useful hints on the appropriateness of the data set provided to the Neural Network and also the set of present training parameters.



● List by size
○ List by patt

```
Patt  Err / Eps                              Cycle 23      163 Errors
006  [6.830]  0.254566 0.498802 0.154755 0.336767 = 1.000000
106  [6.257]  0.214028 0.541579 0.226597 0.386366 = 1.000000
117  [6.201]  0.184536 0.878423 0.401462 0.205990 = 1.000000
015  [6.173]  0.172063 0.880041 0.403195 0.202790 = 1.000000
177  [5.858]  0.102738 0.867444 0.380348 0.201449 = 1.000000
139  [5.744]  0.202417 0.778406 0.538705 0.525341 = 1.000000
190  [5.733]  0.202261 0.746271 0.423220 0.473491 = 1.000000
108  [5.706]  0.182322 0.783022 0.547096 0.535474 = 1.000000
```

TL/DD/12071–18

This window will clearly indicate how many pattern sets indicate error at any specific cycle (iteration) of Neural Network training. One important hint is that if the pattern displaying the maximum error is same after many cycles and also the magnitude of the error is not diminishing, the user may be better off either evaluating the data pattern or resetting the Neural Network training parameters in order to accelerate the training process. More useful hints are provided in the users manual on how to interpret the error window information.

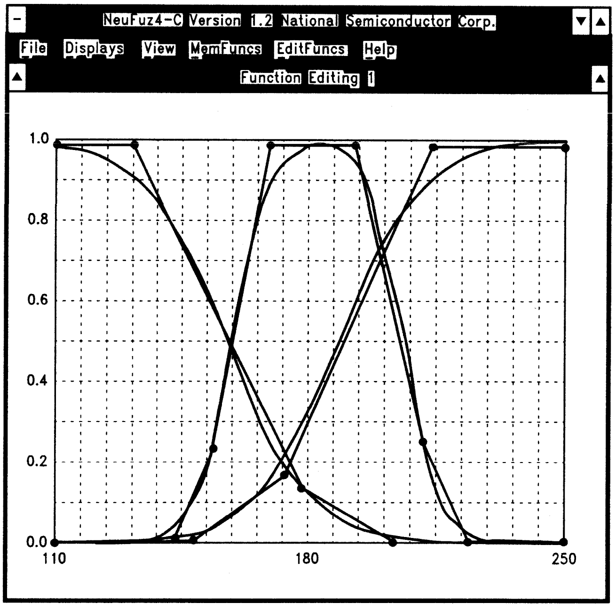
7.5 Fuzzy Rules And Membership Functions Generation

The NeuFuz Neural Network is constructed to map its knowledge directly into Fuzzy Logic. The generated Membership Functions are bell shaped (sigmoidal) as opposed to fixed shapes (triangles, trapezoids and polygons). Bell shaped Membership Functions enhance NeuFuz's ability to represent the knowledge base in Fuzzy Logic. A Rules file is created at this stage, which contains the information of Fuzzy Rules and Membership Functions. NeuFuz generates Rules with crisp output values. For example, a Rule generated by NeuFuz may look like this:

```
"If Temperature is High and Viscosity is
Very Low__then Pump__RPM is 303"
```

7.6 Membership Functions Editing

NeuFuz generates bell shaped Membership Functions. In order to implement the Membership Functions on an embedded processor, the user has the capability to approximate the Membership Functions to a six-vertex polygon (also called a shouldered trapezoid). A graphical window is used to edit the approximated Membership Functions generated by the Neural Network after the training phase. It is necessary to edit the approximations to minimize the error as required by the application.



TL/DD/12071-19

The Rules file is automatically updated when changes to the Membership Function approximations are made.

7.7 FUZZY RULES VERIFICATION AND OPTIMIZATION

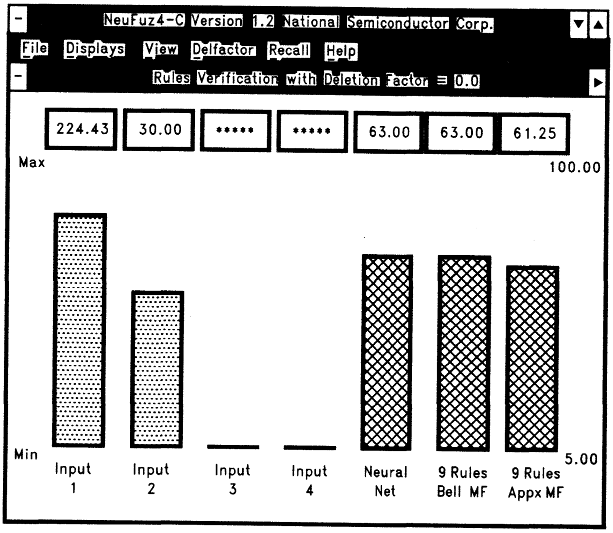
The Rules verification feature allows the user to examine and analyze the three outputs of NeuFuz.

1. Neural Network output.
2. Fuzzy Rules output with bell shaped Membership Functions.
3. Fuzzy Rules output with approximated Membership Functions.

A graphical view of these three outputs can be used to compare the difference in accuracy of the Fuzzy solultions as compared with the Neural Network solution. In addition, the user can use this to check the accuracy of the approximations made to the Membership Functions. The point to note here is that the output of the Fuzzy Rules is the defuzzied value of the contribution of Rules.

7.7.1 Deletion Factor

A deletion factor is used to optimize the number of Fuzzy Logic Rules. Optimization is achieved by eliminating marginal Rules. Having fewer Rules results in smaller output code size and decreases the response time of the Fuzzy Logic solution. The deletion factor is a value between 0 and 1. This value affects the number of Rules in the Rule base and eliminates the less significant Rules. The user has the capability to reevaluate the accuracy of the solution after certain Rules have been eliminated.



TL/DD/12071-20

7.7.2 Recall

NeuFuz has a Recall feature which is used to verify the accuracy of the Fuzzy Logic solution provided by NeuFuz (This is analogous to the recall phase of a Neural Network, when no training occurs). During recall, a set of input patterns is read from a file and an output file is generated with the corresponding outputs from the Neural Network, the Fuzzy Rules with bell-shaped Membership Functions and Fuzzy Rules with approximated Membership Functions, thus providing an easy comparison of these solutions.

7.8 COP8 ASSEMBLY CODE GENERATION

The Code Generation feature is used to generate COP8 Assembly code for the Fuzzy Logic model from the Rules file. The code generator can be used to provide warning messages in case the memory available on the specified COP8 device is exceeded. By clicking on the "RUN CODE GENERATOR" button COP8 assembly code will be automatically generated. This code is richly commented and is extremely compact. This code also includes some general purpose math routines.

7.9 FUZZY EXECUTION MODULE CREATION

In order to create a Fuzzy Execution Module, the Fuzzy Logic Assembly code generated by NeuFuz4 should be integrated with the application routine. This is accomplished by using the COP8 Assembler, linker, debugger and development tools. The following steps are followed by the Fuzzy Execution Module while executing in a COP8 embedded processor.

1. Fuzzification
2. Rule Contribution
3. Rule Evaluation
4. Defuzzification

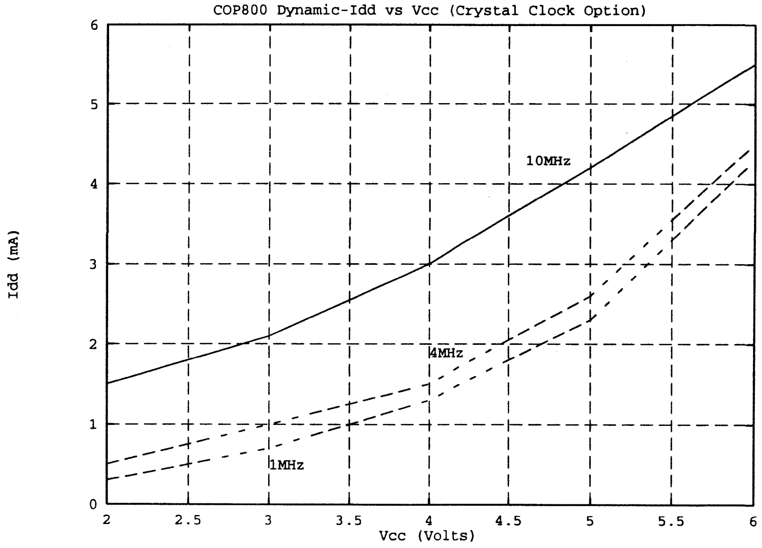
Note: The above mentioned steps of Fuzzification, Rule Contribution, Rule Evaluation and Defuzzification are also performed during Fuzzy Rules Verification and Optimization.

Appendix B

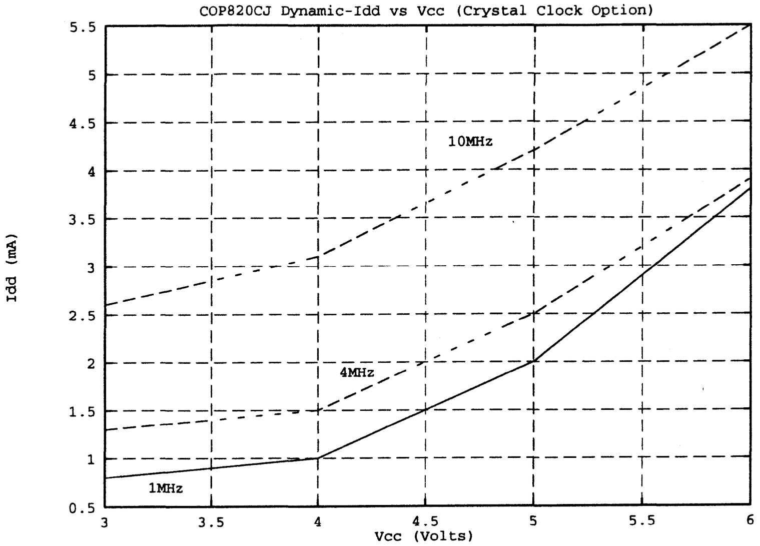
ELECTRICAL CHARACTERIZATION DATA

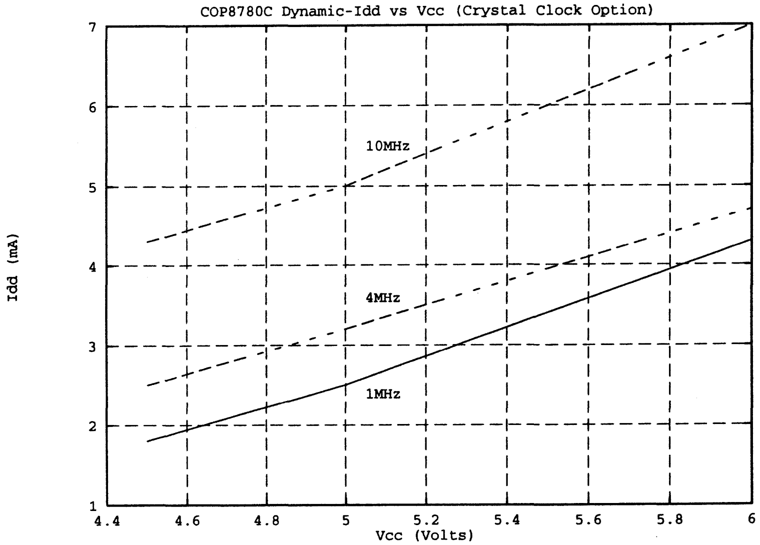
This appendix presents characterization data for the COP800 Basic Family members. All graphs in this appendix apply to the entire COP800 Basic Family unless otherwise noted.

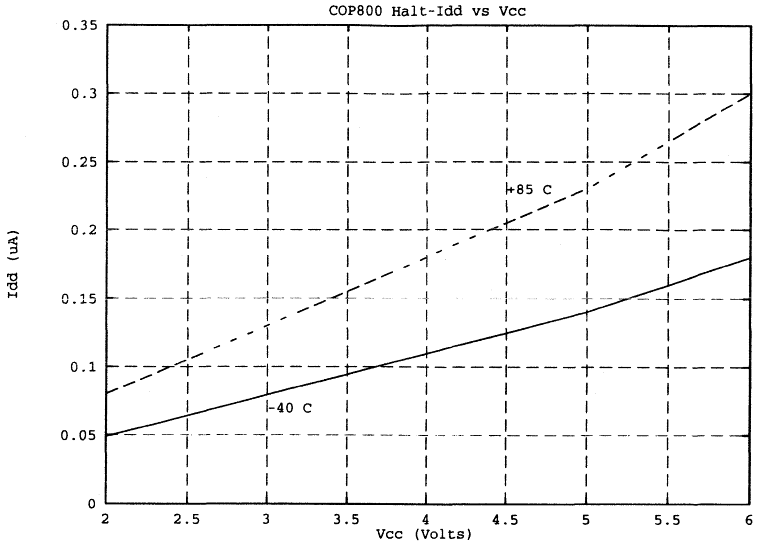
Characterization data is information gained from testing a wide range of sample of parts. Most tests are performed over the full temperature and operating voltage range of the COP800 devices. All information provided in the graphs represents typical values. Most parts will meet these typical values. However, National Semiconductor does not guarantee these values on all parts. Guaranteed numbers are provided in the AC and DC Electrical Characteristics tables found in every datasheet. Guaranteed numbers are tested on all COP800 devices shipped to our customers.



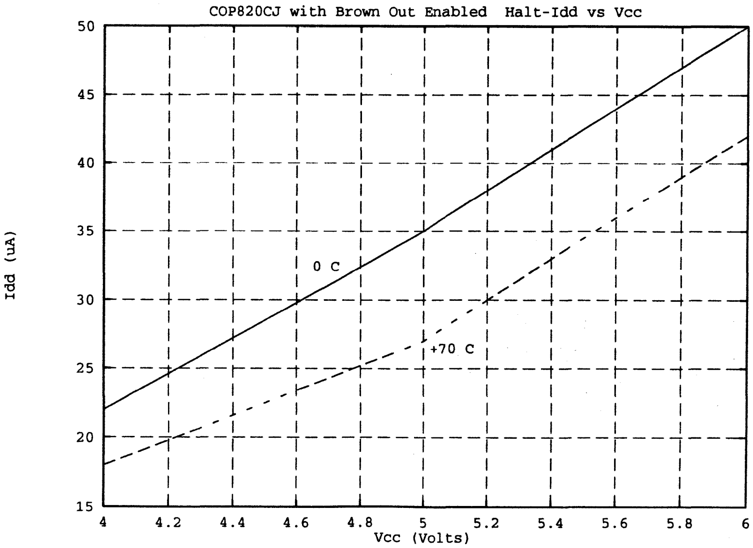
This graph is valid for all COP800 Basic Family members except the COP820CJ and COP8780C.

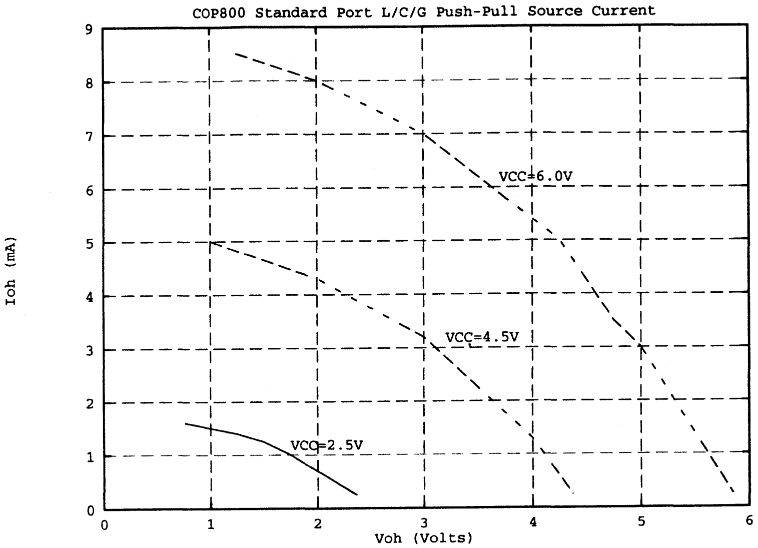

COP820CJ Dynamic-Idd vs Vcc (Crystal Clock Option)
Idd (mA)
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
3
3.5
4
4.5
5
5.5
6
Vcc (Volts)
10MHz
4MHz
1MHz


COP8780C Dynamic-Idd vs Vcc (Crystal Clock Option)
Idd (mA)
7
6
5
4
3
2
1
4.4
4.6
4.8
5
5.2
5.4
5.6
5.8
6
Vcc (Volts)
10MHz
4MHz
1MHz

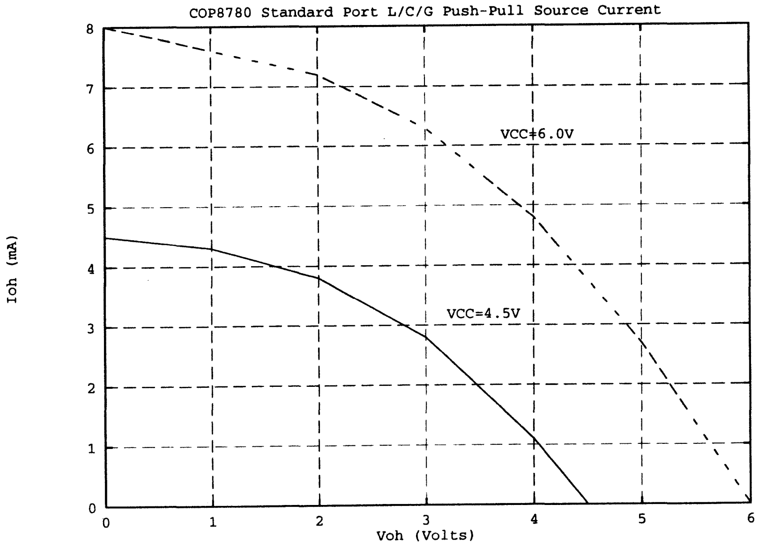


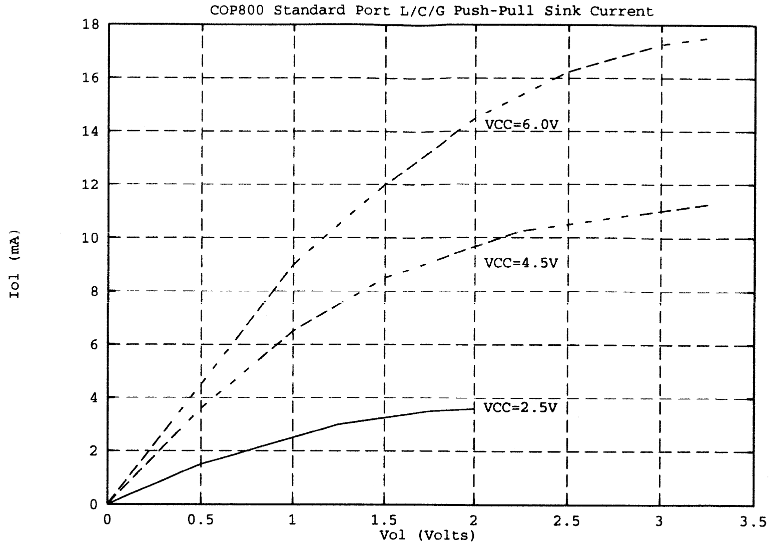
This graph is valid for all COP800 Basic Family members except the COP820CJ with Brown Out enabled.

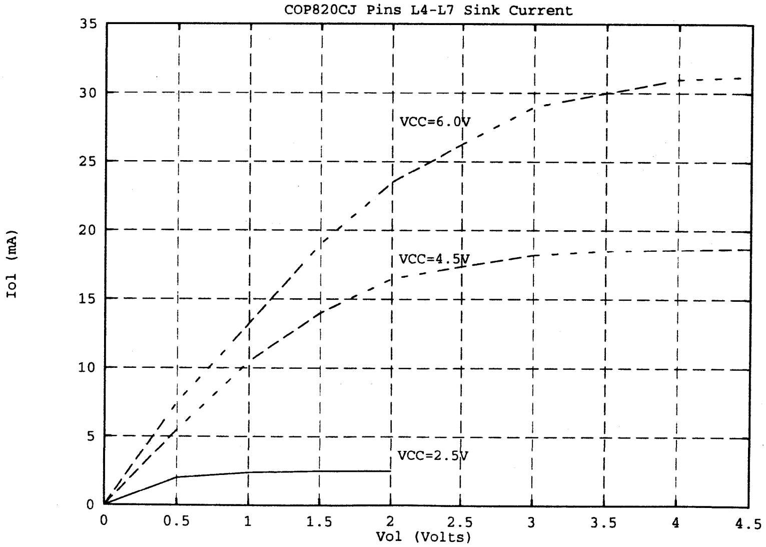


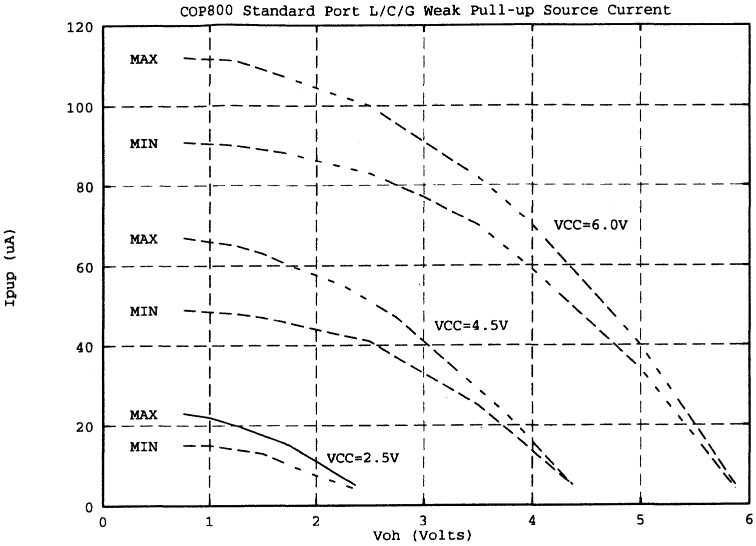


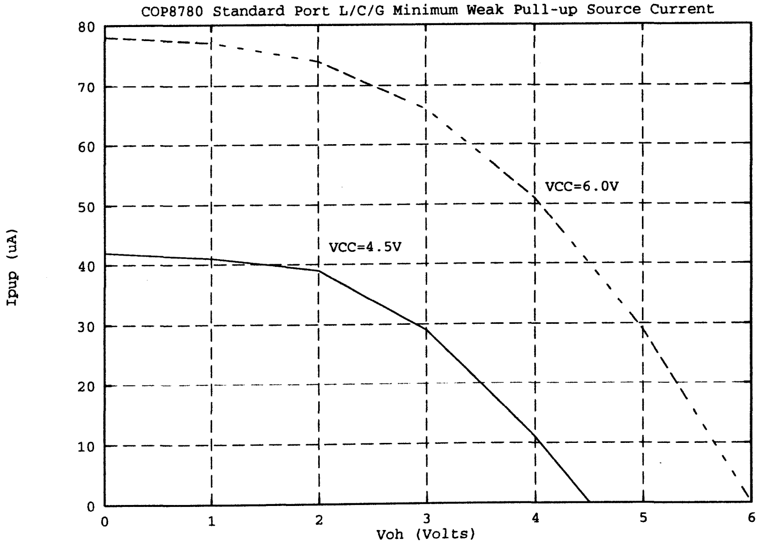
This graph is valid for all COP800 Basic Family members except the COP8780C.

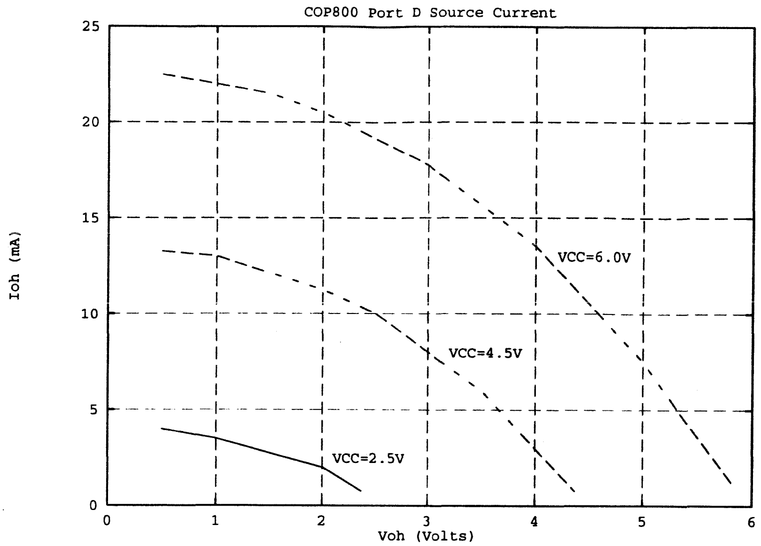



COP800 Standard Port L/C/G Push-Pull Sink Current
Iol (mA)
0
2
4
6
8
10
12
14
16
18
0
0.5
1
1.5
2
2.5
3
3.5
Vol (Volts)
VCC=6.0V
VCC=4.5V
VCC=2.5V

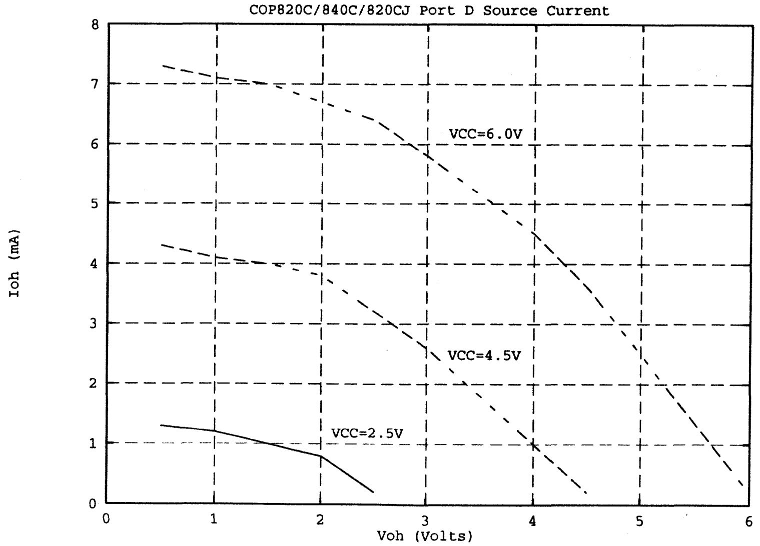

COP820CJ Pins L4-L7 Sink Current
Iol (mA)
0
5
10
15
20
25
30
35
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Vol (Volts)
VCC=6.0V
VCC=4.5V
VCC=2.5V

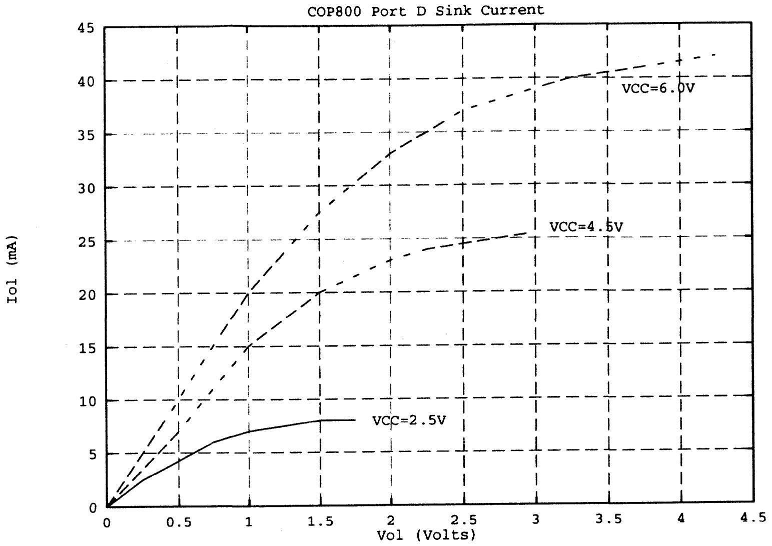

COP800 Standard Port L/C/G Weak Pull-up Source Current
Ipup (uA)
Voh (Volts)
120
100
80
60
40
20
0
0
1
2
3
4
5
6
MAX
MIN
MAX
MIN
MAX
MIN
VCC=6.0V
VCC=4.5V
VCC=2.5V

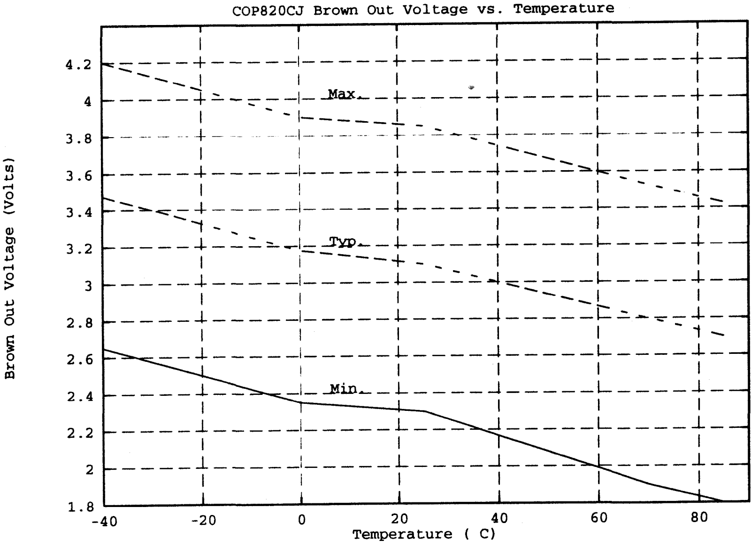

COP8780 Standard Port L/C/G Minimum Weak Pull-up Source Current
Ipup (uA)
Voh (Volts)
80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
VCC=6.0V
VCC=4.5V



This graph is valid for all COP800 Basic Family members except the COP820, COP820CJ and COP840.




COP800 Port D Sink Current
Iol (mA)
45
40
35
30
25
20
15
10
5
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Vol (Volts)
VCC=6.0V
VCC=4.5V
VCC=2.5V


COP820CJ Brown Out Voltage vs. Temperature
Brown Out Voltage (Volts)
4.2
4
3.8
3.6
3.4
3.2
3
2.8
2.6
2.4
2.2
2
1.8
-40
-20
0
20
40
60
80
Temperature (C)
Max.
Typ.
Min.

INDEX