Eight-Bit 80C51 Embedded Processors

1990 Data Book

Advanced Micro Devices

Advanced
Micro
Devices

Eight-Bit 80C51
Embedded Processors

Today Advanced Micro Devices offers you three families of compatible and upgradable CMOS products based on the industry standard 8051 architecture. Our compatible growth path insures that the continuity of your software investment is preserved, not obsoleted.

AMD's microcontrollers increase the levels of performance and reliability of your systems. Our valued-added features are helping set new standards in a variety of telecommunication and computation applications.

The breadth of AMD's portfolio provides maximum flexibility for design and production needs. A variety of program memory densities and fully compatible memory types (EPROM, ROM and ROM-less) are offered in each CMOS family. For example, our EPROM versions greatly simplify prototyping, immediate production starts and rapid code changes.

AMD's extensive worldwide network of sales offices, representatives and distributors is available to provide additional technical support. Please call for assistance today.
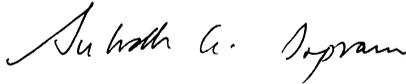
Subodh Toprani
Director of Marketing
Embedded Processor Division

Preface

This databook provides complete information on the wide variety of 8-bit 8051 Family microcontrollers from Advanced Micro Devices. AMD offers not only the best product, but also the necessary documentation and support tools you require.

AMD offers more options on the industry-standard 8051 architecture. Two CMOS product families are now available: the 80C51 and the 80C521. Each family offers a variety of ROM densities and ROM types (on-chip mask ROM, user programmable EPROM, or off-chip ROM). A long list of enhanced features is also available, including Watchdog Timers, Dual Data Pointers, Software Reset, and Port Expansion.

A key to the success of the 8051 Family is the availability of efficient and highly flexible support tools. Excellent emulators, compilers, and programmers are available from multiple sources to meet your requirements. A few of these products are briefly described in Section II. AMD also offers compatible EPROM versions of the 8051 Family to simplify prototyping, initial production, or to provide a tool for immediate program changes.

SECTION I

This section contains general information on the 8051 Family of devices and serves as a core that is useful to designing with all of AMD's microcontrollers. The terms "8051" or "8051 Family" refer to the entire line of 8051-based microcontrollers, each executing an identical instruction set.

SECTION II

This section focuses on specific products, and includes data sheets, device-specific application information and software routines. The data sheets emphasize features unique to the device and do not generally repeat information common to the entire 8051 Family.

Table of Contents

NUMERICAL DEVICE LISTING

SECTION I

8051 Architectural Description

Section 1 presents "core" information applicable to all members of the 8051 Microcontroller Family. In Chapter 1, each member is discussed briefly; an in-depth description of the family's memory organization follows. The information flows naturally into chapters on archi-

tecture, programming, the instruction set, software routines, and Boolean processing capabilities.

As AMD adds more devices to the 8051 Family, this section will continue to serve as a one-stop reference for both hardware and software designers.

Overview

Boolean Processing
Capabilities

Architecture

8051
Family

Software
Routines

Programmer's
Guide

Instruction Set
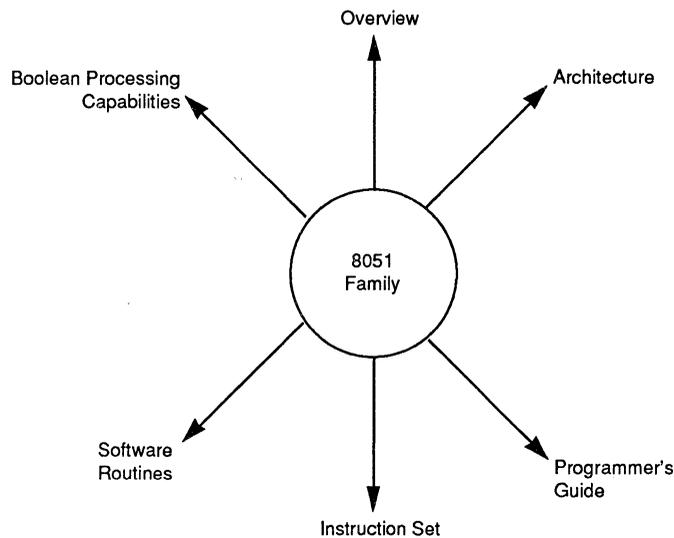
CHAPTER 1

CHAPTER 1
8051 Family Overview

MEMBERS OF THE FAMILY

The 8051 microcontroller family is based upon the architectural structure shown in Figure 1-1. The AMD 80C51 products are shown in Table 1-1.
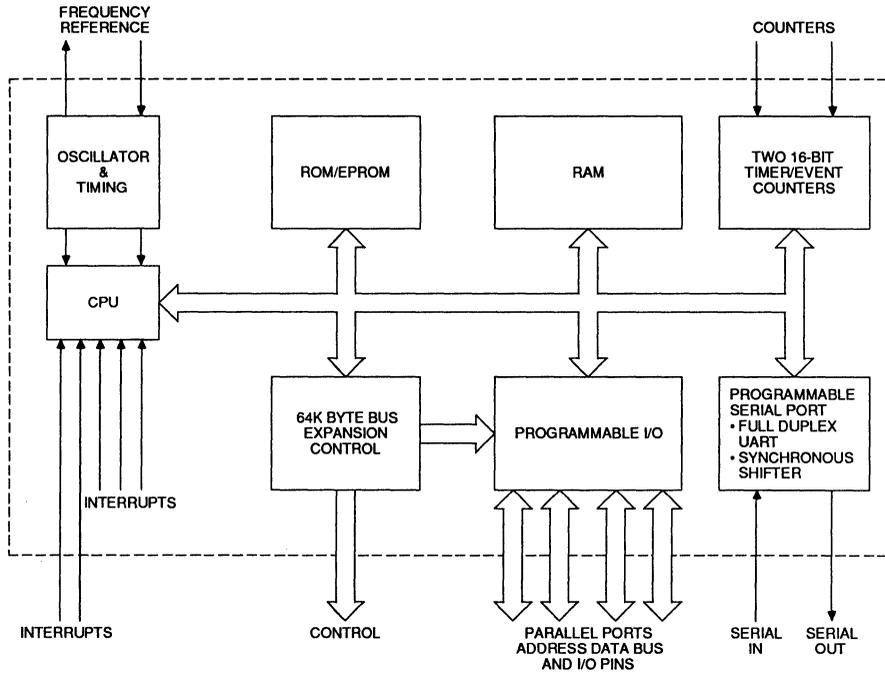
Figure 1-1. Architectural Structure of the 8051 Family

Table 1-1. AMD's 80C51 Family Products

Device	Internal Memory			Timers (16-bit)	Other Enhanced Features
	ROM (bytes)	EPROM (bytes)	RAM (bytes)		
80C31BH	—	—	128	2	—
80C51BH	4K	—	128	2	—
87C51	—	4K	128	2	—
80C32T2	—	—	256	2	—
80C52T2	8K	—	256	2	—
87C52T2	—	8K	256	2	—
80C321	—	—	256	2	Yes
80C521	8K	—	256	2	Yes
87C521	—	8K	256	2	Yes
80C541	16K	—	256	2	Yes
87C541	—	16K	256	2	Yes
80C324	—	—	256	2	Yes
80C325	—	—	256	2	Yes
80C525	8K	—	256	2	Yes
NMOS products are also available.					

80C51BH/80C31BH/87C51

The 80C51BH is a CMOS version of the original NMOS 8051AH offering approximately 80% less power consumption and faster operating speeds. It is fully software compatible with the NMOS device and offers identical features including:

- 8-bit CPU optimized for control applications
- 4K bytes of on-chip Program Memory
- 128 bytes of on-chip Data Memory
- Two 16-bit Timer/Counters
- Full duplex UART
- 5-source interrupt structure with two priority levels
- On-chip oscillator
- Boolean processor
- Bit-addressable RAM
- 64K Program Memory Space
- 64K Data Memory Space

The CMOS product will not always be fully pin-compatible with the NMOS device. Further distinctions between the CMOS and NMOS 8051 Family members may be found in Chapter 7 (Designing with the 80C51BH).

In addition to power savings during normal operation, the 80C51BH offers idle and power-down modes. In idle mode, the CPU is turned off while the RAM and other on-chip peripherals continue to operate. Current draw is typically 15% of the current draw when the device is fully active. In the power-down mode, all on-chip activities are suspended while the RAM holds its data. In this mode, the device typically draws less than 10 µA.

The 80C31BH is identical to the 80C51BH except that it contains no on-chip ROM. The 87C51 is the EPROM version of the 80C51BH. The EPROM products are especially useful for prototyping and immediate production starts.

80C52T2/80C32T2/87C52T2

The 80C52T2 is identical to the 80C51BH except for the amount of on-chip memory. The ROM was increased to 8K bytes and the RAM was increased to 256 bytes. The 80C52T2 has two 16-bit timers similar to the 80C51BH.

The "8052" architecture referred to in this manual is an 8051 with 8K bytes of ROM, 256 bytes of RAM, and a third timer. AMD does not produce an 8052 in either CMOS or NMOS technologies. If the extra timer is not required the 80C52T2 can be used in 8052 applications.

The 80C32T2 is an identical ROM-less version of the 80C52T2. The 87C52T2 is an EPROM version pin-compatible with the 80C52T2.

80C521/80C321/87C521

The 80C521 is an enhanced version of the 80C51. Its additional features include the following:

- 8K bytes of on-chip ROM
- 256 bytes of on-chip RAM
- Programmable Watchdog Timer
- Dual Data Pointers
- Software Reset

The 80C521 is pin-compatible and functional-compatible with the 80C51. The Programmable Watchdog Timer is specially designed to be both flexible and dependable. It provides needed protection from the effects of electrostatic discharge (ESD), external noise, unexpected external events or program anomalies. The dual data pointers facilitate external memory operations such as block moves, saving both time and code space. The 80C321 is the ROM-less version of the 80C521. The 87C521 is the EPROM version of the 80C521.

80C541/87C541

The 80C541 is identical to the 80C521 except the on-chip Program Memory has been increased to 16K bytes. The 87C541 is the EPROM version of the 80C541.

80C324

The 80C324 is a superset of the 80C321 and includes one additional feature—Port Expansion Mode. The 80C324 provides a port expansion capability for adding up to 14 additional full-speed and performance 8-bit I/O ports. The new ports are constructed externally by multiplexing through Port 1 and using EA/PXS for strobe timing. Port 3 operates as normal; however, other ports, including Port 0 and Port 2, which normally are sacrificed for a multiplexed data/address bus, are reconstructed.

The new ports are accessed by software exactly as if they existed on-chip. The entire 8051 family instruction set is available for these additional ports. Traditional memory-mapped I/O ports allow only four instructions to be used, vastly reducing their effectiveness.

MEMORY ORGANIZATION IN 8051 FAMILY DEVICES

Logical Separation of Program and Data Memory

All 8051 Family devices have separate address spaces for Program and Data Memory, as shown in Figure 1-2.

The logical separation of Program and Data Memory allows the Data Memory to be accessed by 8-bit addresses, which can be more quickly stored and manipulated by an 8-bit CPU. Nevertheless, 16-bit Data Memory addresses can also be generated through the DPTR register.

Program Memory can only be read, not written to. There can be up to 64K bytes of Program Memory. In the 80C51BH and the 87C51, the lowest 4K bytes of Program Memory are on-chip. The read strobe for external Program Memory is the signal $\overline{PSEN}$ (Program Store Enable).

Data Memory occupies a separate address space from Program Memory. Up to 64K bytes of external RAM can be addressed in the external Data Memory space. The CPU generates read and write signals, $\overline{RD}$ and $\overline{WR}$ as needed during external Data Memory accesses.

External Program Memory and external Data Memory may be combined if desired by applying the $\overline{RD}$ and $\overline{PSEN}$ signals to the inputs of an AND gate and using the output of the gate as a read strobe to the external Program/Data Memory.

Program Memory

Figure 1-3 shows a map of the lower part of Program Memory. After reset, the CPU begins execution from location 0000H.

As shown in Figure 1-3, each interrupt is assigned a fixed location in Program Memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose Program Memory.

Interrupt service locations are spaced at 8-byte intervals: 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the

PROGRAM MEMORY
(READ ONLY)

DATA MEMORY
(READ/WRITE)
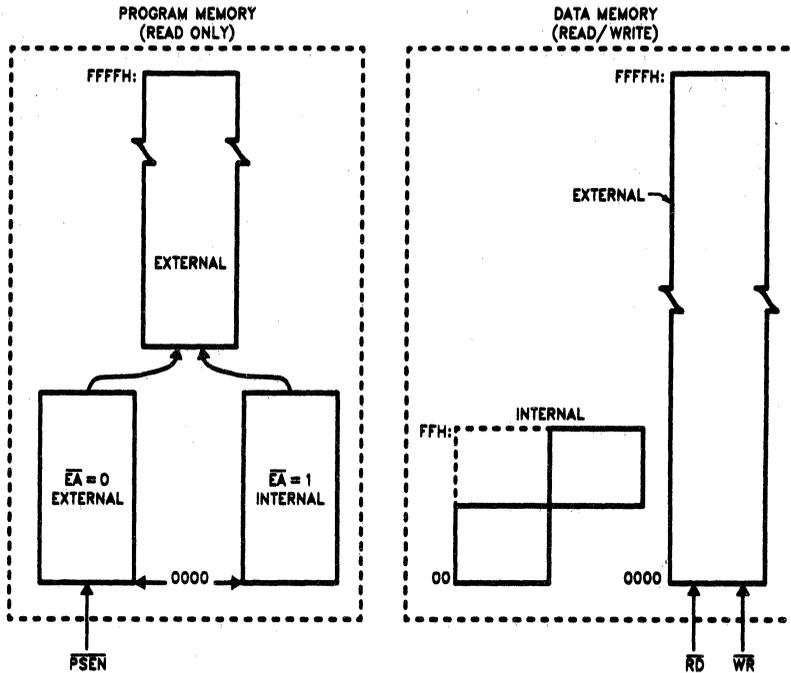
Figure 1-2. 80C51 Memory Structure

case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupt locations are in use.

The lowest 4K (or 8K in the 80C52T2/80C521) bytes of Program Memory can be either in the on-chip ROM or in an external ROM. This selection is made by strapping the $\overline{EA}$ (External Access) pin to either V_{cc} or V_{ss}.

In the 80C51, if the $\overline{EA}$ pin is strapped to V_{cc}, then program fetches to addresses 0000H through 0FFFH are directed to the internal ROM. Program fetches to addresses 1000H through FFFFH are directed to external ROM.

In the 80C52T2/80C521, $\overline{EA} = V_{cc}$ selects addresses 0000H through 1FFFH to be internal, and addresses 2000H through FFFFH to be external.

If the $\overline{EA}$ pin is strapped to V_{ss}, then all program fetches are directed to external ROM. The ROMless parts must have this pin externally strapped to V_{ss} to enable them to execute from external Program Memory.

The read strobe to external ROM, $\overline{PSEN}$, is used for all external program fetches. $\overline{PSEN}$ is not activated for internal program fetches.

The hardware configuration for external program execution is shown in Figure 1-4. Note that 16 I/O lines (Ports 0 and 2) are dedicated to bus functions during external Program Memory fetches. Port 0 (P0 in Figure 1-4) serves as a multiplexed address/data bus. It emits the low byte of the Program Counter (PCL) as an address, and then goes into a float state awaiting the arrival of the
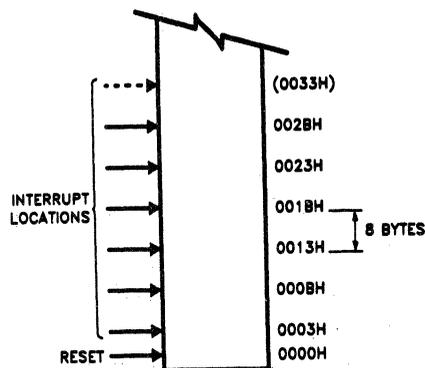
Figure 1-3. 80C51 Program Memory

code byte from the Program Memory. During the time that the low byte of the Program Counter is valid on P0, the signal ALE (Address Latch Enable) clocks this byte into an address latch. Meanwhile, Port 2 (P2 in Figure 1-4) emits the high byte of the Program Counter (PCH). Then $\overline{\text{PSEN}}$ strobes the EPROM and the code byte is read into the microcontroller.

Program Memory addresses are always 16 bits wide, even though the actual amount of Program Memory used may be less than 64K bytes. External Program execution sacrifices two of the 8-bit ports, P0 and P2, to the function of addressing the Program Memory.

Data Memory

The right half of Figure 1-2 shows the internal and external Data Memory spaces available to the 8051 Family user.

Figure 1-5 shows a hardware configuration for accessing up to 2K bytes of external RAM. The CPU in this case is executing from internal ROM. Port 0 serves as a multiplexed address/data bus to the RAM, and 3 lines of Port 2 are being used to page the RAM. The CPU generates $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals as needed during external RAM accesses.

There can be up to 64K bytes of external Data memory. External Data Memory addresses can be either 1 or 2 bytes wide. One-byte addresses are often used in conjunction with one or more other I/O lines to page the RAM, as shown in Figure 1-5. Two-byte addresses can also be used, in which case the high address byte is emitted at Port 2.

Internal Data Memory is mapped in Figure 1-6. The memory space is shown divided into three blocks, which are generally referred to as the Lower 128, the Upper 128, and SFR space.

Internal Data Memory addresses are always 1 byte wide, which implies an address space of only 256 bytes. However, the addressing modes for internal RAM can in fact accommodate 384 bytes, using a simple trick. Direct addresses higher than 7FH access one memory space, and indirect addresses higher than 7FH access a different memory space. Thus Figure 1-6 shows the Upper 128 and SFR space occupying the same block of addresses, 80H through FFH, although they are physically separate entities.

The Lower 128 bytes of RAM are present in all 8051 Family devices as mapped in Figure 1-7. The lowest 32 bytes are grouped into 4 banks of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) select which register bank is in use. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing.

The next 16 bytes above the register banks form a block of bit-addressable memory space. The 8051 Family instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH.

All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing. The Upper 128 (Figure 1-8) can only be accessed by indirect addressing. The Upper 128 bytes of RAM are not implemented in the 80C51.

Figure 1-9 gives a brief look at the Special Function Register (SFR) space. SFRs include the Port latches,
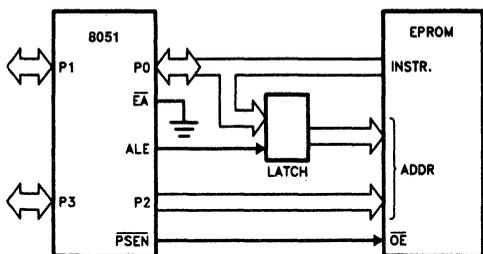
Figure 1-4. Executing from External Program Memory
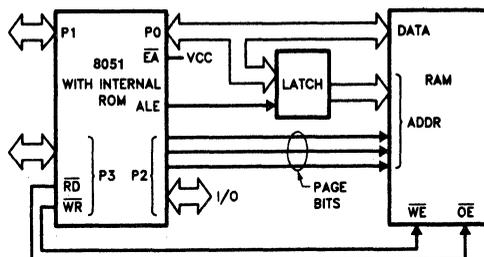
Figure 1-5. Accessing External Data Memory.
If the Program Memory is Internal, the Other
Bits of P2 are Available as I/O
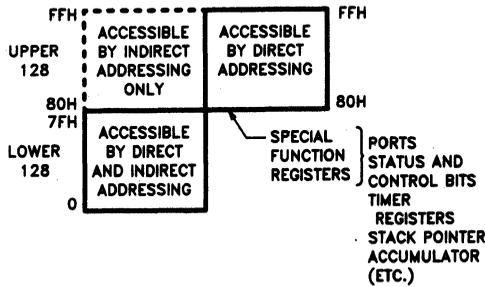
Figure 1-6. Internal Data Memory
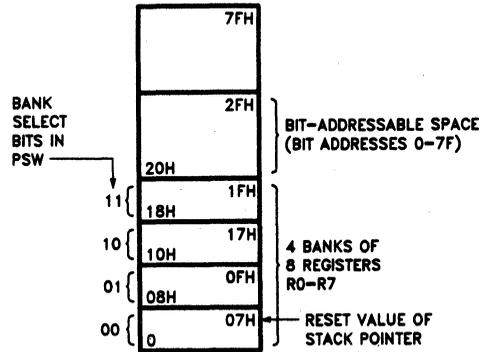
Figure 1-7. The Lower 128 Bytes of Internal RAM

timers, peripheral controls, etc. These registers can only be accessed by direct addressing. In general, all 8051 Family microcontrollers have the same SFRs as the 80C51, and at the same addresses in SFR space. However, enhancements to the 80C51 have additional SFRs that are not present in the 80C51, nor perhaps in other proliferations of the family.

Sixteen addresses in SFR space are both byte- and bit-addressable. The bit-addressable SFRs are those whose address ends in 000B. The bit addresses in this area are 80H through FFH.
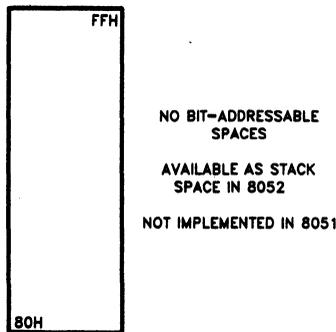
Figure 1-8. The Upper 128 Bytes of Internal RAM
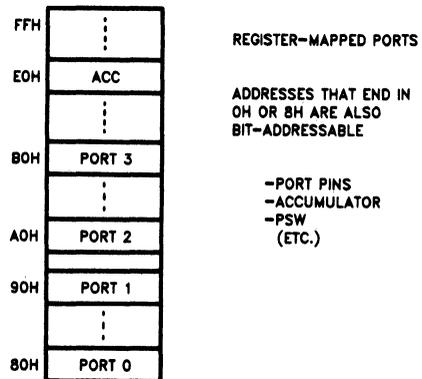
Figure 1-9. SFR Space

CHAPTER 2

CHAPTER 2
8051 Family Architecture

INTRODUCTION

This chapter and the remainder of Section I covers the basic architecture and instruction set of the 8051 Family. In these chapters the terms "8051" and "8051 Family" refer to the entire family of microcontrollers in both CMOS and NMOS technologies. Differences in functionality between the CMOS and NMOS products will be specifically noted where they occur.

The term "8052" refers to a version of the 8051 with double the amount of memory (8K bytes ROM and 256

bytes RAM) and an extra timer. In this section it will be used specifically to describe changes due to this third timer.

Section II focuses on AMD's portfolio of CMOS 80C51 products. It is organized by product family with datasheets, application notes and other information pertaining to features beyond the basic core architecture described in Section I. Thus, the reader experienced with the 8051 may wish to begin in Section II.
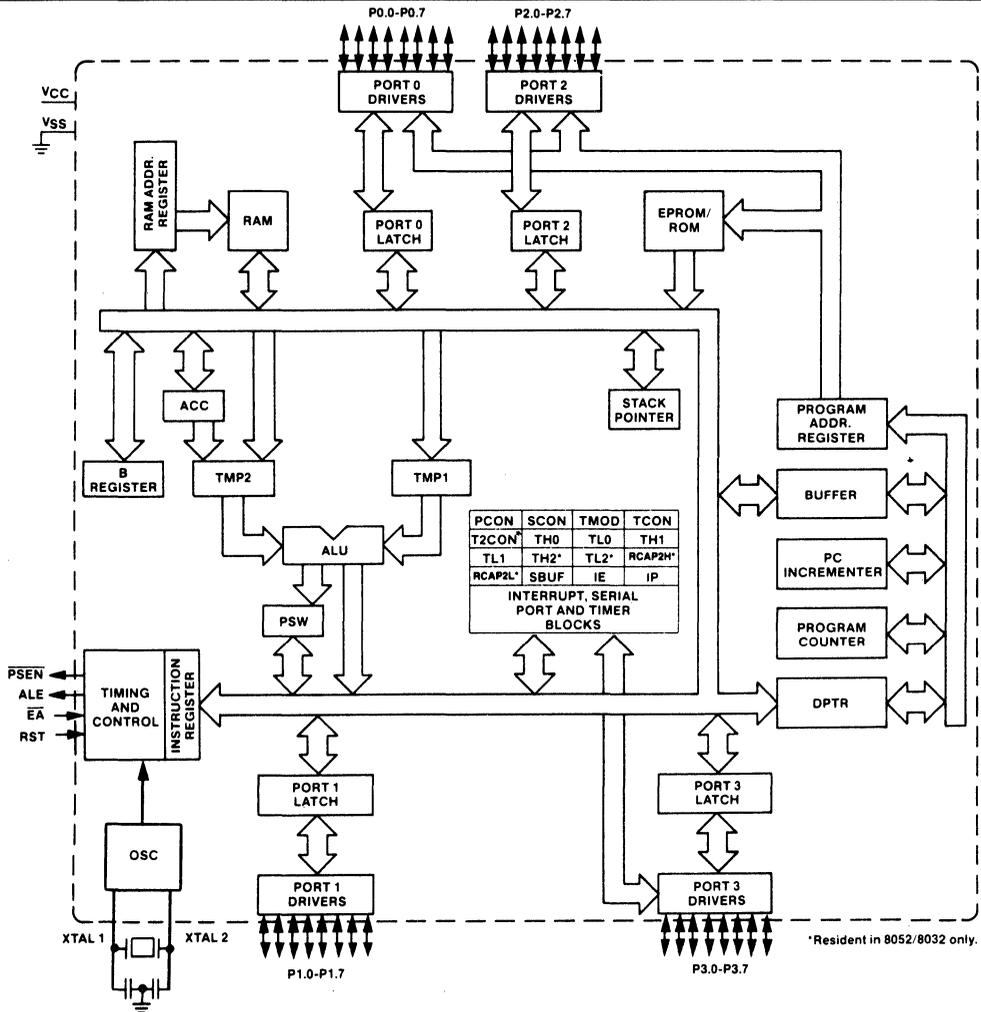
Figure 2-1. 8051 Family Architecture

Table 2-1 80C51 Core Products

Part	Internal Memory		
	ROM (bytes)	EPROM (bytes)	RAM (bytes)
80C31BH	-	-	128
80C51BH	4K	-	128
87C51	-	4K	128
80C32T2	-	-	256
80C52T2	8K	-	256
87C52T2	-	8K	256

The major 8051 Family features are:

- 8-bit CPU
- On-Chip oscillator and clock circuitry
- 32 I/O lines
- 64K bytes address space for external Data Memory
- 64K bytes address space for external Program Memory
- Two 16-bit timer/counters (three on 8032/8052)
- A five-source interrupt structure (six sources on 8032/8052) with two priority levels
- Full duplex serial port
- Boolean Processor

MEMORY ORGANIZATION

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K bytes (8K for 80C52T2) may reside on-chip. The Data Memory can consist of up to 64K bytes of off-chip RAM, in addition to which it includes 128 bytes of on-chip RAM (256 bytes for the 80C52T2), plus a number of "SFRs" (Special Function Registers) as listed below.

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer (consisting of DPH and DPL)	83H 82H
*P0	Port 0	80H
*P1	Port 1	90H

Symbol	Name	Address
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
+ *T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 (high byte)	8CH
TL0	Timer/Counter 0 (low byte)	8AH
TH1	Timer/Counter 1 (high byte)	8DH
TL1	Timer/Counter 1 (low byte)	8BH
+ TH2	Timer/Counter 2 (high byte)	0CDH
+ TL2	Timer/Counter 2 (low byte)	0CCH
+ RCAP2H	Timer/Counter 2 Capture Register (high byte)	0CBH
+ RCAP2L	Timer/Counter 2 Capture Register (low byte)	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buff	99H
PCON	Power Control	87H

The SFRs marked with an asterisk (*) are both bit- and byte-addressable. The SFRs marked with a plus sign (+) are present in timer 2 of the 8052 only. The functions of the SFRs are described as follows.

Accumulator

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B Register

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

Program Status Word

The PSW register contains program status information as detailed in Figure 2-2.

(MSB)						(LSB)	
CY	AC	F0	RS1	RS0	OV	—	P

Symbol	Position	Name and Significance	Symbol	Position	Name and Significance
CY	PSW.7	Carry flag.	—	PSW.1	(reserved)
AC	PSW.6	Auxiliary Carry flag. (For BCD operations.)	P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.
F0	PSW.5	Flag 0 (Available to the user for general purposes.)			
RS1	PSW.4	Register bank Select control bits 1 & 0.			
RS0	PSW.3	Set/cleared by software to determine working register bank (see Note).	Note—		the contents of (RS1, RS0) enable the working register banks as follows:
OV	PSW.2	Overflow flag.			

(0.0)—Bank 0   (00H-07H)
(0.1)—Bank 1   (08H-0FH)
(1.0)—Bank 2   (10H-17H)
(1.1)—Bank 3   (18H-1FH)

Figure 2-2. PSW: Program Status Word Register

Stack Pointer

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

Data Pointer

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

Ports 0 to 3

P0, P1, P2, and P3 are the SFR latches of Ports 0, 1, 2, and 3, respectively.

Serial Data Buffer

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

Timer Registers

Register pairs (TH0, TL0), (TH1, TL1), and (TH2, TL2) are the 16-bit counting registers for Timer/Counters 0, 1, and 2, respectively.

Capture Registers

The register pair (RCAP2H, RCAP2L) are the capture registers for the Timer 2 "capture mode." In this mode, in

response to a transition at the 8052's T2EX pin, TH2 and TL2 are copied into RCAP2H and RCAP2L. Timer 2 also has a 16-bit auto-reload mode, and RCAP2H and RCAP2L, hold the reload value for this mode. More about Timer 2's features on page 2-12.

Control Registers

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the timer/counters, and the serial port. They are described in later sections.

OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the output and input of a single-stage on-chip inverter, which can be configured with off-chip components as a Pierce oscillator, as shown in Figure 2-3. The on-chip circuitry, and selection of off-chip components to configure the oscillator are discussed on page 2-28.

30pf ± 10pf FOR CRYSTALS
40pf ± 10pf FOR CERAMIC RESONATORS

30pf ± 10pf FOR CRYSTALS
40pf ± 10pf FOR CERAMIC RESONATORS
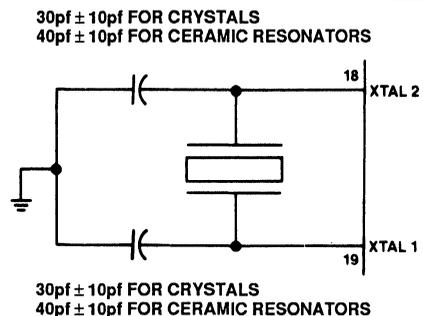
Figure 2-3. Crystal/Ceramic Resonator Oscillator

The oscillator drives the internal clock generator, which provides the internal clocking signals to the chip. The internal clocking signals are at half the oscillator frequency, and define the internal phases, states, and machine cycles, described in the next section.

CPU TIMING

A machine cycle consists of six states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half, during which the Phase 2 clock is active. Thus, a machine cycle consists of 12 oscillator periods, numbered S1P1
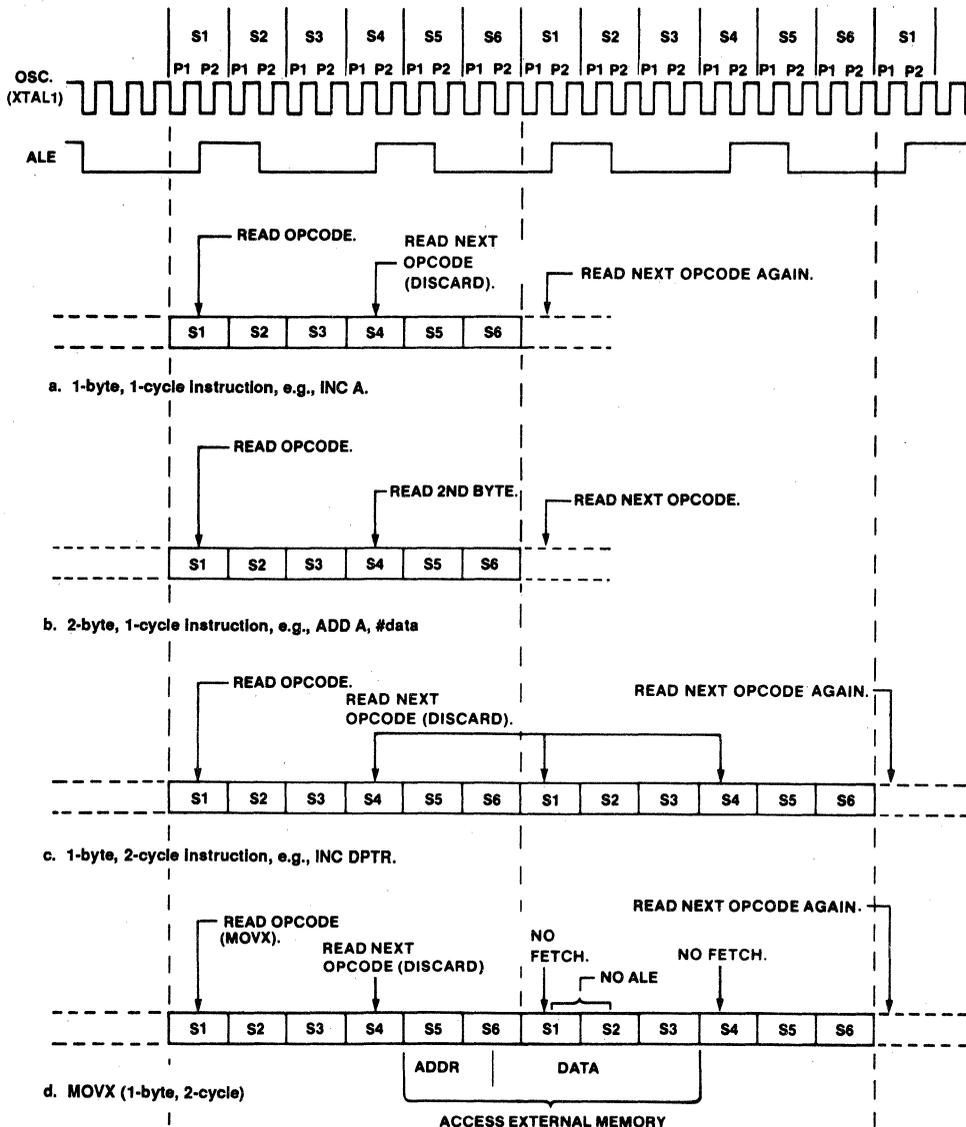
Figure 2-4. 8051 Fetch/Execute Sequences

(State 1, Phase 1) through S6P2 (State 6, Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. Typically, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 2-4 show the fetch/execute timing referenced to the internal states and phases. Since these internal clock signals are not user accessible, the XTAL1 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: one during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a 2-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a 1-byte instruction, there is still a fetch at S4, but the byte read (which would be the next opcode) is ignored, and the Program Counter is not incremented. In any case, execution is complete at the end of S6P2. Figures 2-4a and 2-4b show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete. They take four cycles.

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a 1-byte 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 2-4c and 2-4d show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external

a. Port 0 Bit

b. Port 1 Bit

c. Port 2 Bit

d. Port 3 Bit
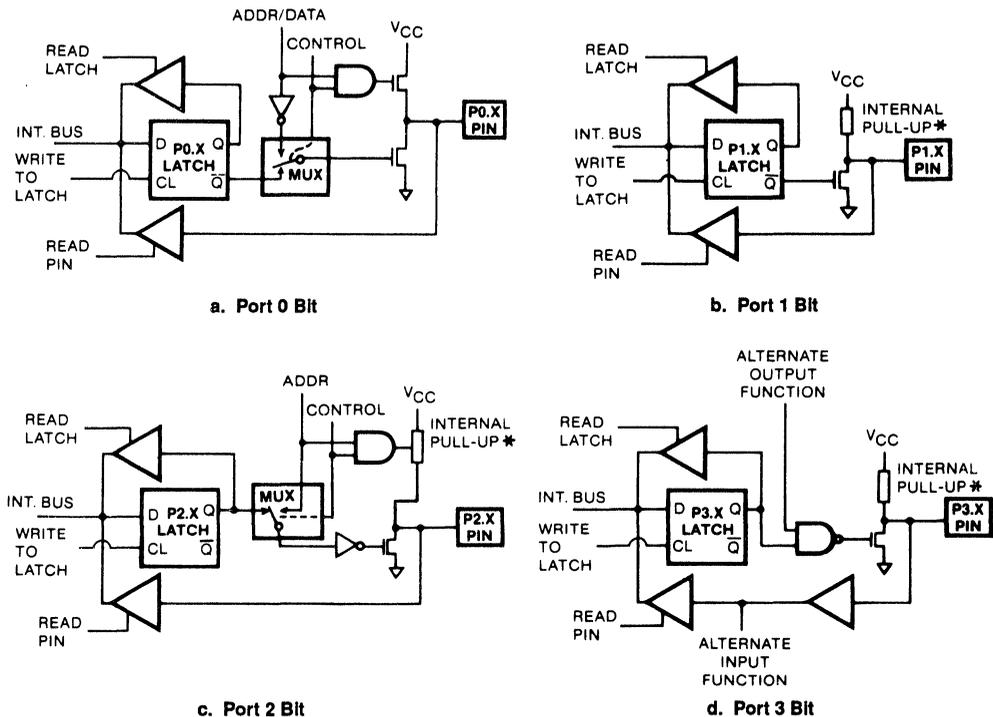
*See Figure 2-6 for details of the internal pull up.

Figure 2-5. 8051 Port Bit Latches and I/O Buffers

memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and (in the 8052) two Port 1 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed below:

Port Pin	Alternate Function
*P1.0	T2 (Timer/Counter 2 external input)
*P1.1	T2EX (Timer/Counter 2 capture/reload trigger)
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{WR}$ (external Data memory write strobe)
P3.7	$\overline{RD}$ (external Data memory read strobe)

*P1.0 and P1.1 serve these alternate functions only on the 8052.

The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

I/O Configurations

Figure 2-5 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a Type D flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The Q output of the flip-flop is placed on the internal bus in response to a "read latch" signal from the CPU. The level of the port pin itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal, and others activate the "read pin" signal. More about that on page 2-8.

As shown in Figure 2-5, the output drivers of Ports 0 and 2 are switchable to an internal ADDR and ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the P2 SFR remains unchanged, but the P0 SFR gets 1s written to it.

Also shown in Figure 2-5, is that if a P3 bit latch contains a 1, then the output level is controlled by the signal labeled "alternate output functions." The actual P3.X pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS.) To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source.

Port 0 differs in not having internal pullups. The pullup FET in the P0 output driver (see Figure 2-5a) is used only when the Port is emitting 1s during external memory accesses. Otherwise the pullup FET is off. Consequently P0 lines that are being used as output port lines are open drain. Writing a 1 to the bit latch leaves both output FETs off, so the pin floats. In that condition it can be used as a high-impedance input.

Because Ports 1, 2, and 3 have fixed internal pullups they are sometimes called "quasi-bidirectional" ports. When configured as inputs they pull high and will source current (IIL, in the data sheets) when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional, because when configured as an input it floats.

All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1.) Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can. It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangements are shown in Figure 2-6.

In NMOS versions of the 8051, the fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source about 0.25 mA when shorted to ground. In parallel with the fixed pull-up is an enhancement-mode transistor, which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.

In the CMOS versions, the pull-up consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when logical 1 is applied to its gate, and is turned off when a logical 0 is applied to its gate. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

Transistor pFET 1 in Figure 2-6 is turned on for two oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET 3 (a weak pull-up) through the inverter. This inverter and pFET 3 form a latch which holds the 1.

Note that if the pin is emitting a 1, a negative glitch on the pin from some external source can turn off pFET 3, causing the pin to go into a float state; pFET 2 is a very weak pull-up which is on whenever the nFET is off, in traditional CMOS style. It's only about 1/10 the strength of pFET 3. Its function is to restore a 1 to the pin in the event the pin *had* a 1 and lost it to a glitch.
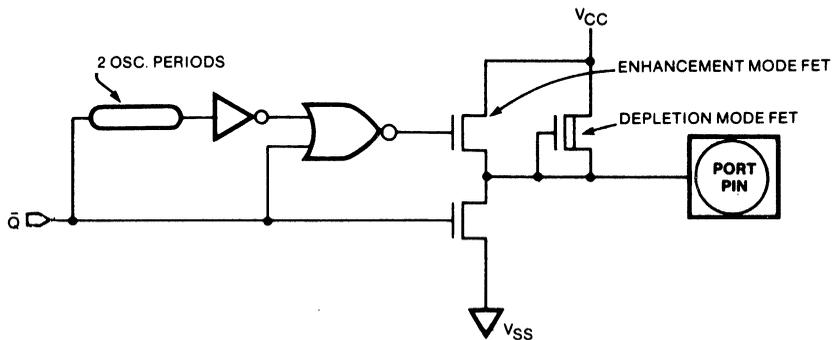
a. NMOS Configuration
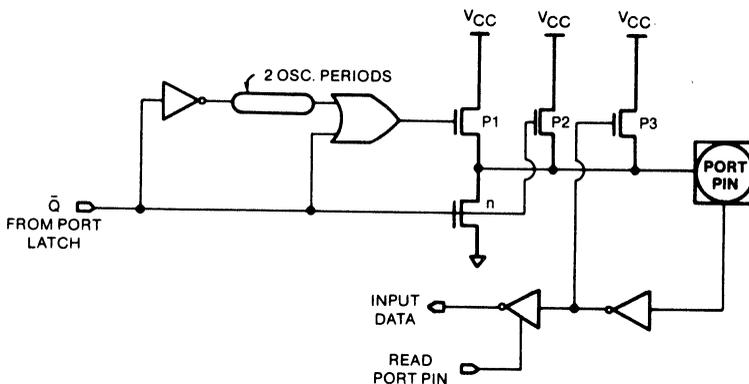
b. CMOS Configuration

Figure 2-6. Ports 1 and 3 NMOS and CMOS Internal Pull-up Configurations.
(Port 2 is similar except that it holds the strong pull-up on while emitting 1s that are address bits.)

Port Loading and Interfacing

The output buffers of Ports 1, 2, and 3 can each drive four LS TTL inputs. The ports on NMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both NMOS and CMOS pins can be driven by open-collector and open-drain outputs, but note that 0-to-1 transitions will not be fast. In the NMOS device, if the pin is driven by an open-collector output, a 0-to-1 transition will have to be driven by the relatively weak depletion mode FET in Figure 2-6a. In the CMOS device, an input 0 turns off pull-up pFET3, leaving only the very weak pull-up pFET2 to drive the transition.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

Read-Modify-Write Feature

Some instructions that read a port, also read the latch, and others read the pin. Which ones do which? The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions, listed below. When the destination operand is a port or a port bit, these instructions read the latch rather than the pin:

ANL	(logical AND, e.g., ANL P1,A)
ORL	(logical OR, e.g., ORL P2,A)
XRL	(logical EX-OR, e.g., XRL P3,A)
JBC	(jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL)
CPL	(complement bit, e.g., CPL P3.0)
INC	(increment, e.g., INC P2)
DEC	(decrement, e.g., DEC P2)
DJNZ	(decrement and jump if not zero, e.g., DJNZ P3, LABEL)
MOV PX.Y,C	(move carry bit to bit Y of Port X)
CLR PX.Y	(clear bit Y of Port X)
SET PX.Y	(set bit Y of Port X)

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal $\overline{PSEN}$ (program store enable) as the read strobe. Accesses to external Data Memory use $\overline{RD}$ or $\overline{WR}$ (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address (MOVX @DPTR) or an 8-bit address (MOVX @Ri).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. Note that the Port 2 drivers use the strong pull-ups during the entire time that they are emitting address bits that are 1s. This is during the execution of a MOVX @DPTR instruction. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used (MOVX @Ri), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and do not require external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before $\overline{WR}$ is activated, and remains there until after $\overline{WR}$ is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

1) Whenever signal $\overline{EA}$ is active; or
2) Whenever the program counter (PC) contains a number that is larger than 0FFFH (1FFFH for the 80C52T2)

This requires that the ROMless versions have $\overline{EA}$ wired low to enable the lower 4K (8K for the 80C32T2) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC. During this time the Port 2 drivers use the strong pull-ups to emit PC bits that are 1s.

PSEN

The read strobe for external fetches is $\overline{PSEN}$, which is not activated for internal fetches. When the CPU is access-

ing external Program Memory, $\overline{PSEN}$ is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When $\overline{PSEN}$ is activated, its timing is not the same as $\overline{RD}$. A complete $\overline{RD}$ cycle, including activation and deactivation of ALE and $\overline{RD}$, takes 12 oscillator periods. A complete $\overline{PSEN}$ cycle, including activation and deactivation of ALE, and $\overline{PSEN}$, takes 6 oscillator periods. The execution sequence for these two types of read cycles is shown in Figure 2-7 for comparison.

ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external Program Memory. For that purpose ALE is activated twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn't come out is during an access to external
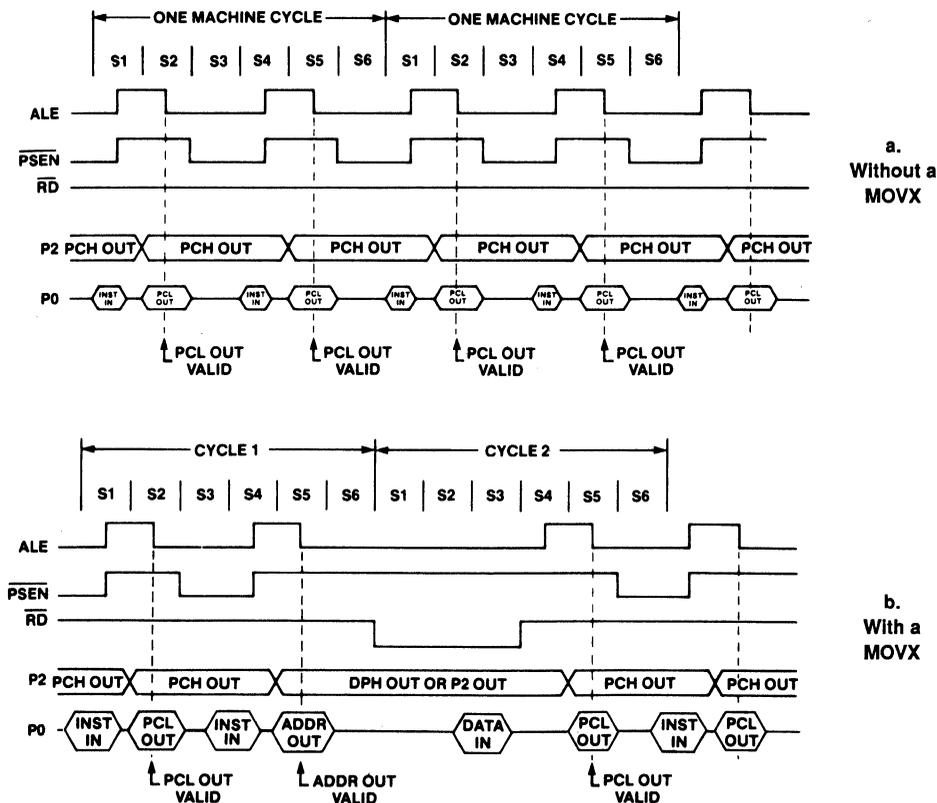
Figure 2-7. External Program Memory Execution

Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 2-7). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.

Overlapping External Program and Data Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051, the external Program and Data Memory spaces can be combined by ANDing $\overline{PSEN}$ and $\overline{RD}$. A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the $\overline{PSEN}$ cycle is faster than the $\overline{RD}$ cycle, the external memory needs to be fast enough to accommodate the $\overline{PSEN}$ cycle.

TIMER/COUNTERS

The 8051 has two 16-bit timer/counter registers: Timer 0 and Timer 1. The 8052 has these two plus one more: Timer 2. All three can be configured to operate either as timers or event counters.

In the "timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding exter-

nal input pin, T0, T1, or (in the 8052) T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "timer" or "counter" selection, Timer 0 and Timer 1 have four operating modes from which to select. Timer 2, in the 8052, has three modes of operation: "capture," "auto-reload" and "baud rate generator."

Timer 0 and Timer 1

These timer/counters are present in both the 8051 and the 8052. The "timer" or "counter" function is selected by control bits C/$\overline{T}$ in the Special Function Register TMOD (Figure 2-8). These two timer/counters have four operating modes, which are selected by bit-pairs (M1, M0) in counters. Mode 3 is different. The four operating modes are described below.

Mode 0

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divided-by-32 prescaler. Figure 2-9 shows the mode 0 operation as it applies to Timer 1.

(MSB)							(LSB)
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0

TIMER 1   TIMER 0

GATE	Gating Control when set. Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control pin is set. When cleared Timer "x" is enabled whenever "TRX" control bit is set.		
C/T	Timer or Counter Selector Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).		

M1	M0	Operating Modes.
0	0	8048 TIMER "TLx" serves as 5-bit prescaler.
0	1	16-bit Timer/Counter "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer-counter "THx" hold a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an 8-bit timer-counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer-counter 1 stopped.

Figure 2-8 TMOD: Timer/Counter Mode Control Register

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Figure 2-10). GATE is in TMOD.

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and INT0 for the corresponding Timer 1 signals in Figure 2-9. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3)

Mode 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

Mode 2

Mode 2 configures the timer register as an 8-bit counter (TL1) with automatic reload, as shown in Figure 2-11. Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

Mode 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.
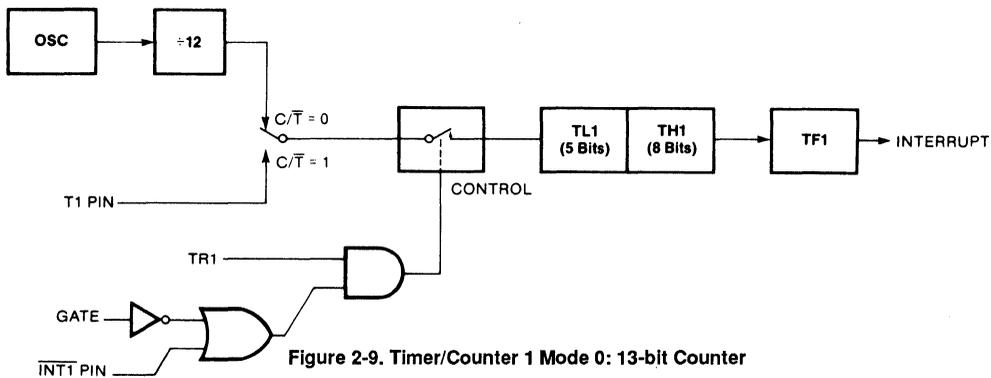
Figure 2-9. Timer/Counter 1 Mode 0: 13-bit Counter

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance	Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.	IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.	IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.	IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.	IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

Figure 2-10. TCON: Timer/Counter Control Register

Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 2-12. TL0 uses the Timer 0 control bits: C/T̄, GATE, TR0, INT0, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. With Timer 0 in Mode 3, an 8051 can look like it has three timer/counters, and an 8052, like it has four. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

Timer 2

Timer 2 is a 16-bit timer/counter which is present only in the 8052. Like Timers 0 and 1, it can operate either as a timer or as an event counter. This is selected by bit C/T̄2 in the Special Function Register T2CON (Figure 2-13). It has three operating modes: "capture," "autoLoad" and
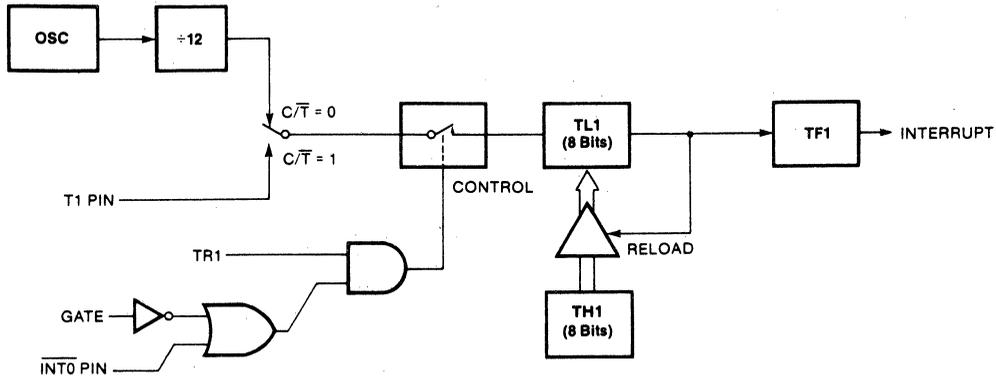
Figure 2-11. Timer/Counter 1 Mode 2: 8-bit Auto-Reload
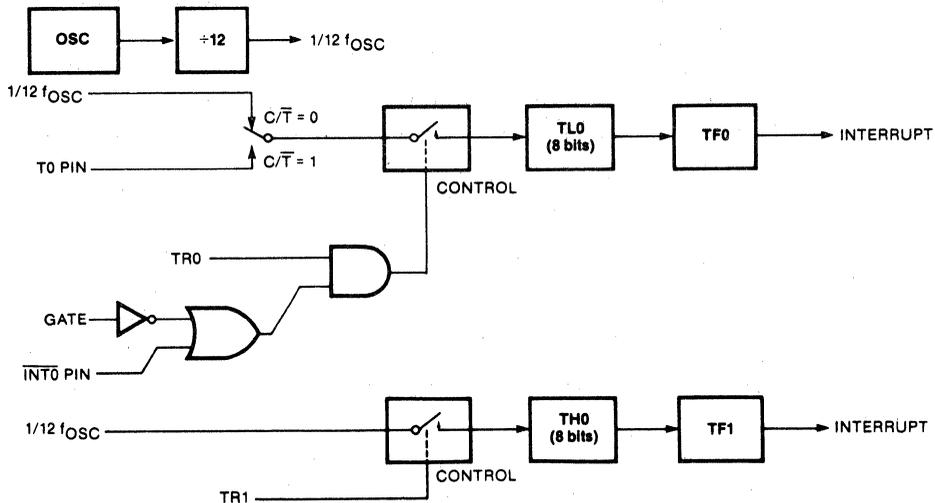
Figure 2-12. Timer/Counter 0 Mode 3: Two 8-bit Counters

(MSB)							(LSB)
TF2	EXF2	RCLK	TCLK	EXEN2	TR	C/$\overline{T2}$	CP/$\overline{RL2}$

Symbol	Position	Name and Significance
TF2	T2CON.7	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.
RCLK	T2CON.5	Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	T2CON.4	Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	T2CON.2	Start/stop control for Timer 2. A logic 1 starts the timer.

C/$\overline{T2}$	T2CON.1	Timer or counter select (Timer 2)
		    0 = Internal timer (OSC/12)
		    1 = External event counter (falling edge triggered).

CP/$\overline{RL2}$	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Figure 2-13. T2CON: Timer/Counter 2 Control Register.

"baud rate generator" which are selected by bits in T2CON as shown in Table 2-2.

Table 2-2. Timer 2 Operating Modes

RCLK + TCLK	CP/$\overline{RL2}$	TR2	MODE
0	0	1	16-bit auto-reload
0	1	1	16-bit capture
1	X	1	baud rate generator
X	X	0	(off)

In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter which upon overflowing sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The capture mode is illustrated in Figure 2-14.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 2-15.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at
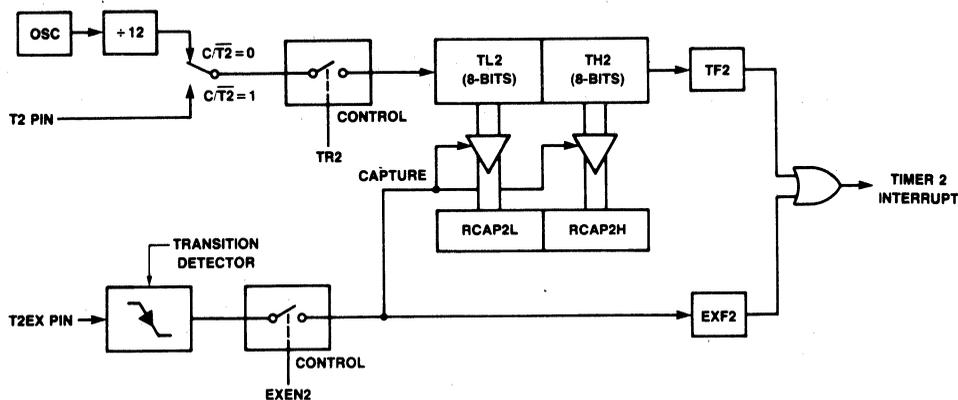
Figure 2-14. Timer 2 in Capture Mode

Special Function Register SBUF. Writing to SBUF loads the transmit register, and reading SBUF accesses a physically separate receive register.

The serial port can operate in 4 modes:

Mode 0: Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Mode 1: 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

Mode 2: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

Mode 3: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

Serial Port Control Register

The serial port control and status is the Special Function Register SCON, shown in Figure 2-16. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (T1 and R1).
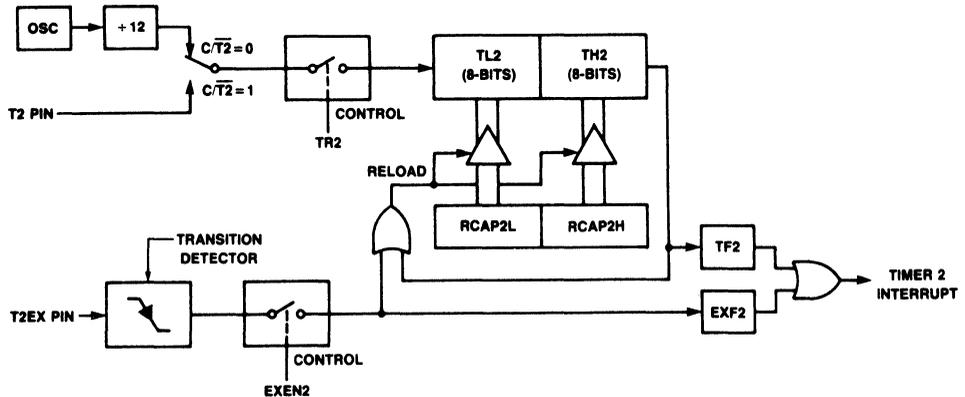
Figure 2-15. Timer 2 in Auto-Reload Mode

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

where SM0, SM1 specify the serial port mode, as follows:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	^{f}osc./12
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	^{f}osc./64 or ^{f}osc./32
1	1	3	9-bit UART	variable

• SM2 enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.

• REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.

• TB8 is the 9th data bit that will be transmitted in modes 2 and 3. Set or clear by software as desired.

• RB8 in modes 2 and 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.

• TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.

• RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

Figure 2-16. SCON: Serial Port Control Register

Baud Rates

The baud rate in Mode 0 is fixed:

$$\text{Mode 0 Baud Rate} = \frac{\text{Oscillator Frequency}}{12}$$

The baud rate in Mode 2 depends on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value on reset), the baud rate is 1/64 the oscillator frequency. If SMOD = 1, the baud rate is 1/32 the oscillator frequency.

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{Oscillator Frequency})$$

In the 8051, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit and the other for receive).

Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

Modes 1, 3
$$\text{Baud Rate} = \frac{2^{SMOD}}{32} \times (\text{Timer 1 Overflow Rate})$$

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either "timer" or "counter" operation, and in any of its 3 running modes. In the most typical applications, it is configured for "timer" operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator Frequency}}{12 \times [256 - (TH1)]}$$

One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD = 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

Figure 2-17 lists various commonly used baud rates and how they can be obtained from Timer 1

Baud Rate	f_{osc}	SMOD	C/$\overline{T}$	Mode	Reload Value
Mode 0 MAX: 1MHz	12 MHz	X	X	X	X
Mode 2 MAX: 375K	12 MHz	1	X	X	X
Modes 1, 3: 62.5K	12 MHz	1	0	2	FFH
19.2K	11.059 MHz	1	0	2	FDH
9.6K	11.059 MHz	0	0	2	FDH
4.8K	11.059 MHz	0	0	2	FAH
2.4K	11.059 MHz	0	0	2	F4H
1.2K	11.059 MHz	0	0	2	E8H
137.5K	11.986 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

Figure 2-17. Timer 1 Generated Commonly Used Baud Rates

Using Timer 2 to Generate Baud Rates

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Figure 2-13). Note then the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 2-18.
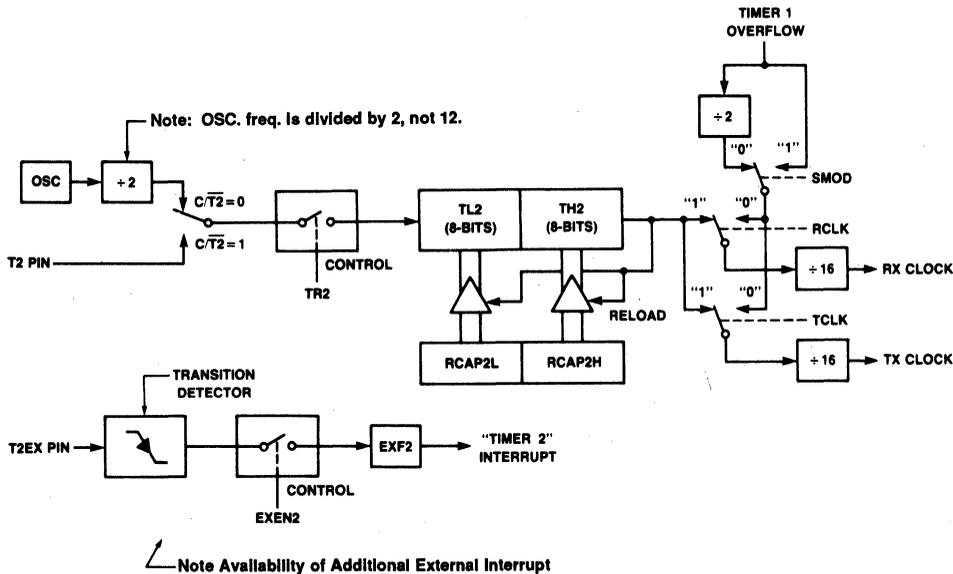
Figure 2-18. Timer 2 in Baud Rate Generator Mode

The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

Now, the baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The Timer can be configured for either "timer" or "counter" operation. In the most typical applications, it is configured for "timer" operation ($C/\overline{T2} = 0$). "Timer" operation is a little different for Timer 2 when it's being used as a baud rate generator. Normally as a timer it would increment every machine cycle (thus at 1/1 the oscillator frequency). As a baud rate generator, however, it increments every state time (thus at 1/2 the oscillator frequency). In that case the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Oscillator Frequency}}{32 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 2-18. This Figure is valid only if RCLK + TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt, if desired.

It should be noted that when Timer 2 is running (TR2 = 1) in "timer" function in the baud rate generator mode, one should not try to read or write TH2 or TL2. Under these conditions the Timer is being incremented every state time, and the results of a read or write may not be accurate. The RCAP registers may be read, but shouldn't be written to, because a write might overlap a reload and cause write and/or reload errors. Turn the Timer off (clear TR2) before accessing the Timer 2 or TCAP registers, in this case.

More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Figure 2-19 shows a simplified functional diagram of the serial port in mode 0, and associated timing.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF," and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeros. This condition flags the TX Control block to do one last shift and then deactivate SEND and set T1. Both of these actions occur at S1P1 of the 10th machine cycle after "write to SBUF."

Reception is initiated by the condition REN = 1 and RI = 0. At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.
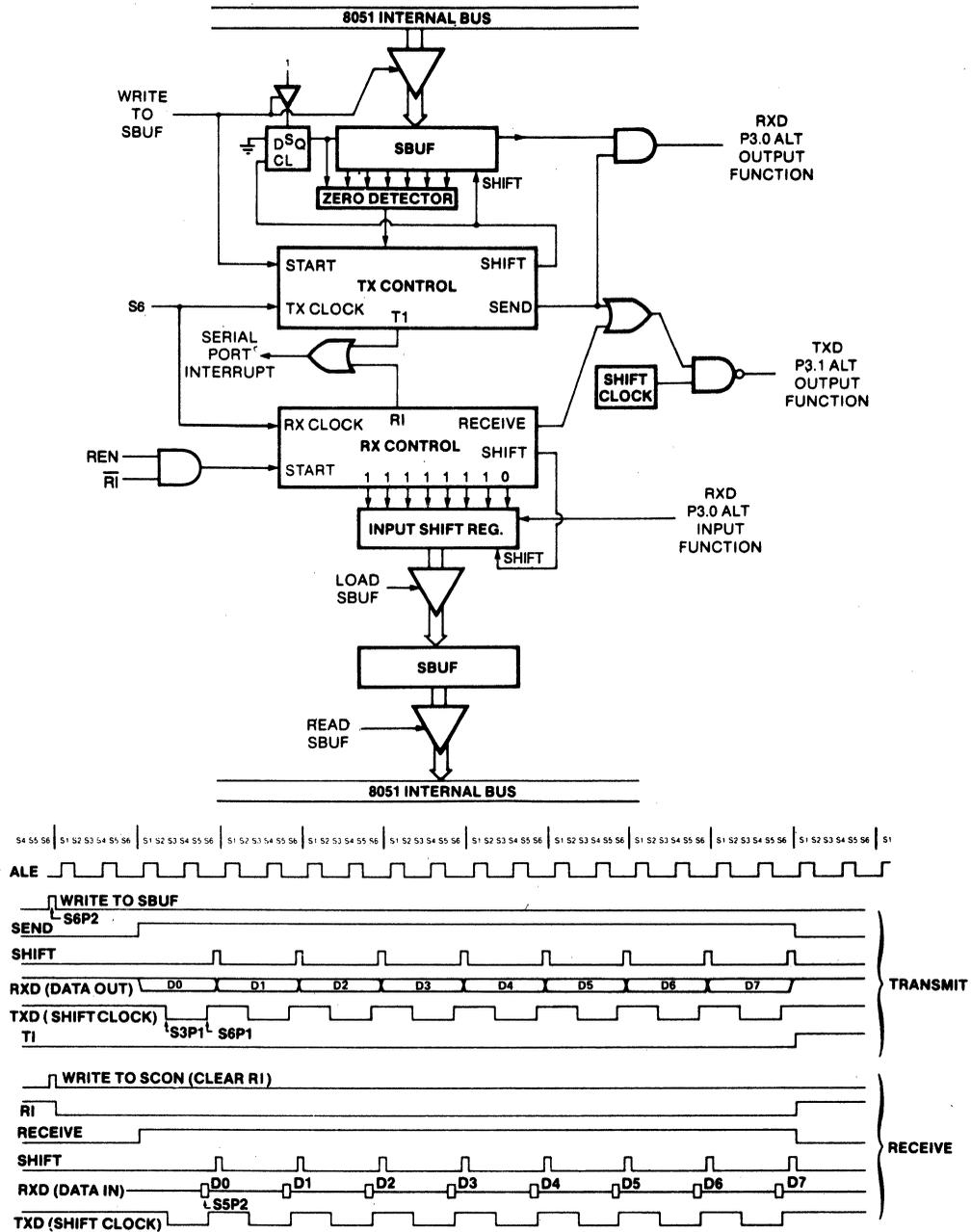
Figure 2-19. Serial Port Mode 0

More About Mode 1

Ten bits transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. In the 8051 the baud rate is determined the the Timer 1 overflow rate. In the 8052 it is determined either by the Timer 1 overflow rate, or the Timer 2 overflow rate, or both (one for transmit and the other for receive).

Figure 2-20 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit and receive.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of $\overline{\text{SEND}}$, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeros are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate $\overline{\text{SEND}}$ and set TI. This occurs at the 10th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

1) RI = 0, and
2) Either SM2 = 0, or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

More About Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2. Mode 8 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figures 21a and b show a functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in. Thus, as data bits shift out to the
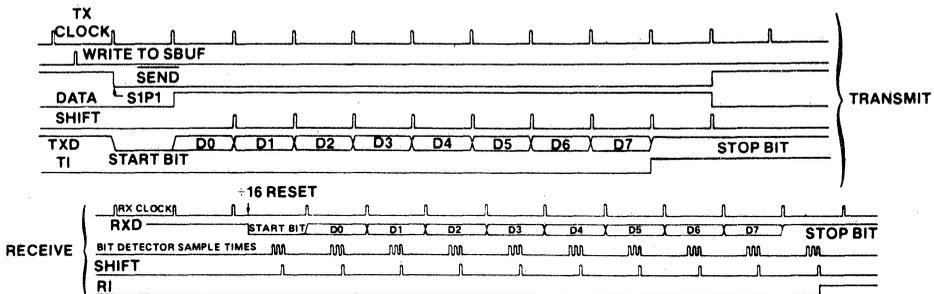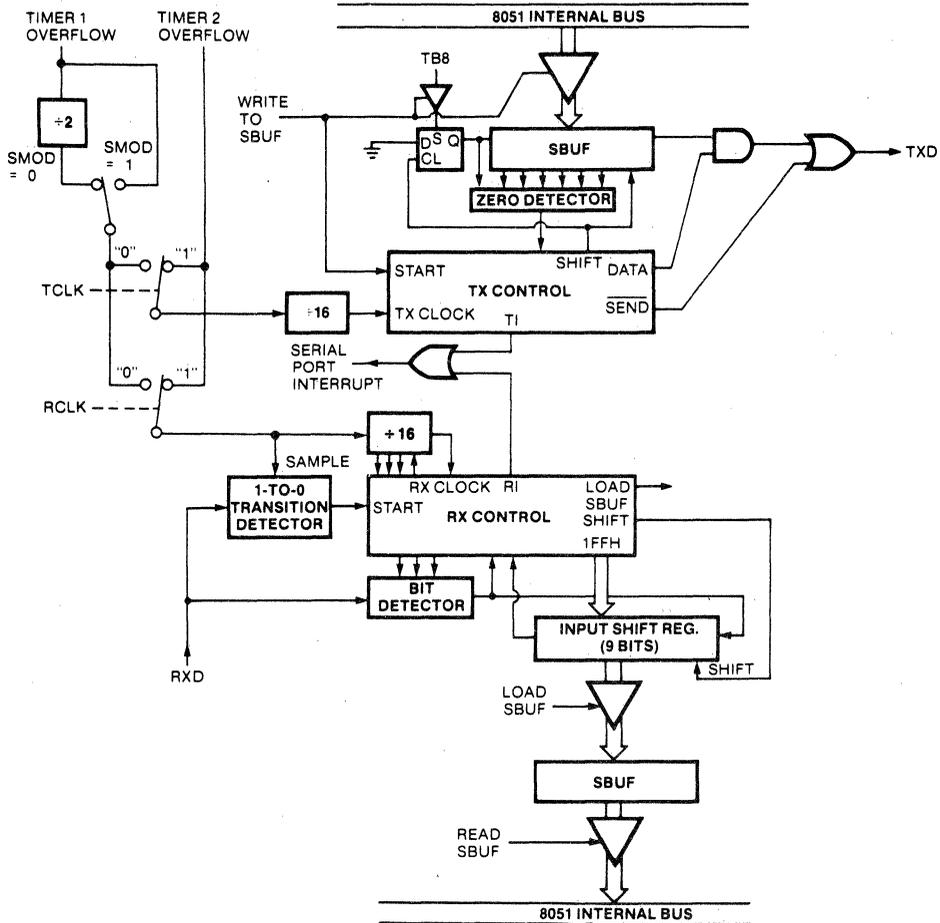
Figure 2-20. Serial Port Mode 1
(TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.)
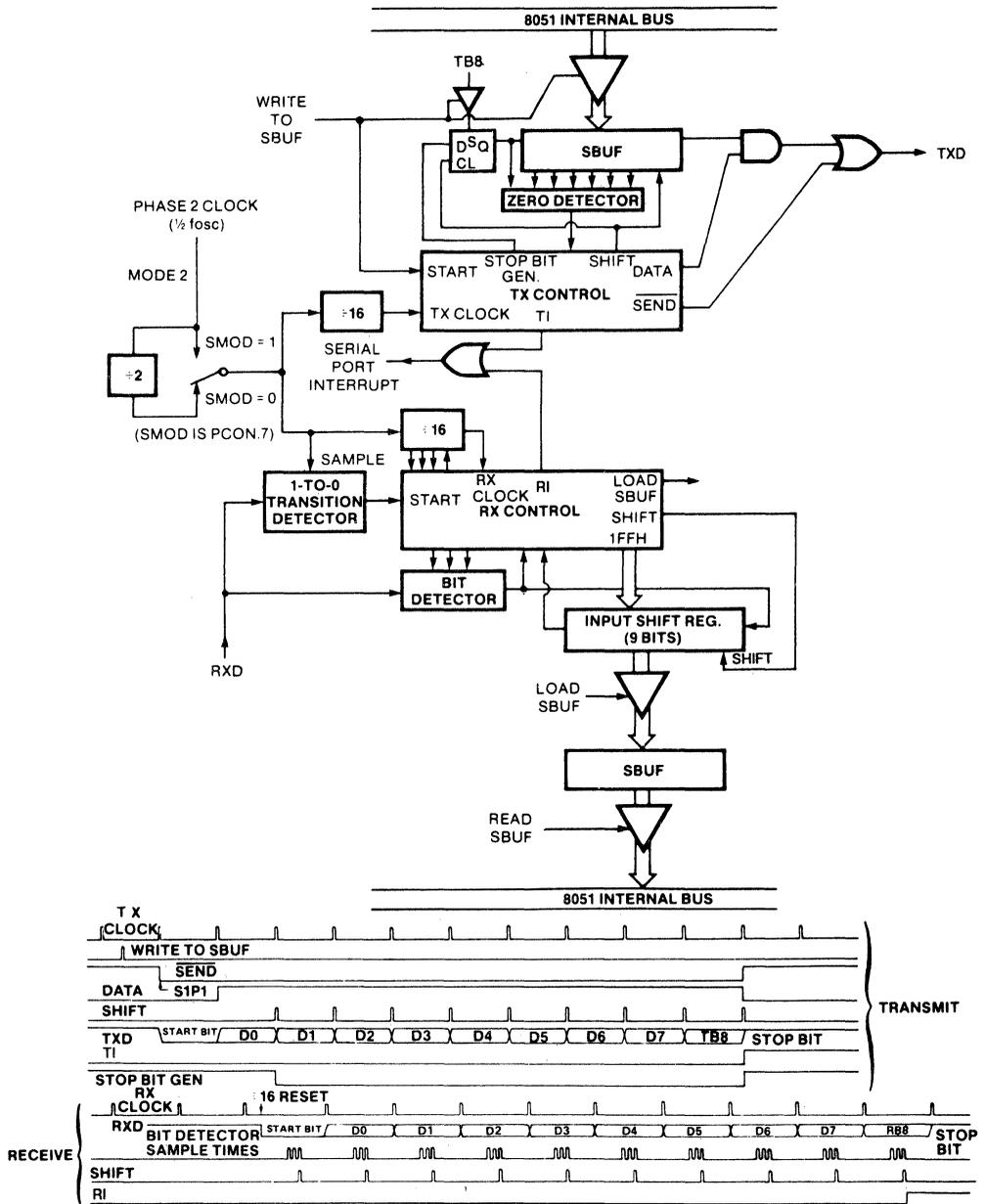
Figure 2-21a. Serial Port Mode 2
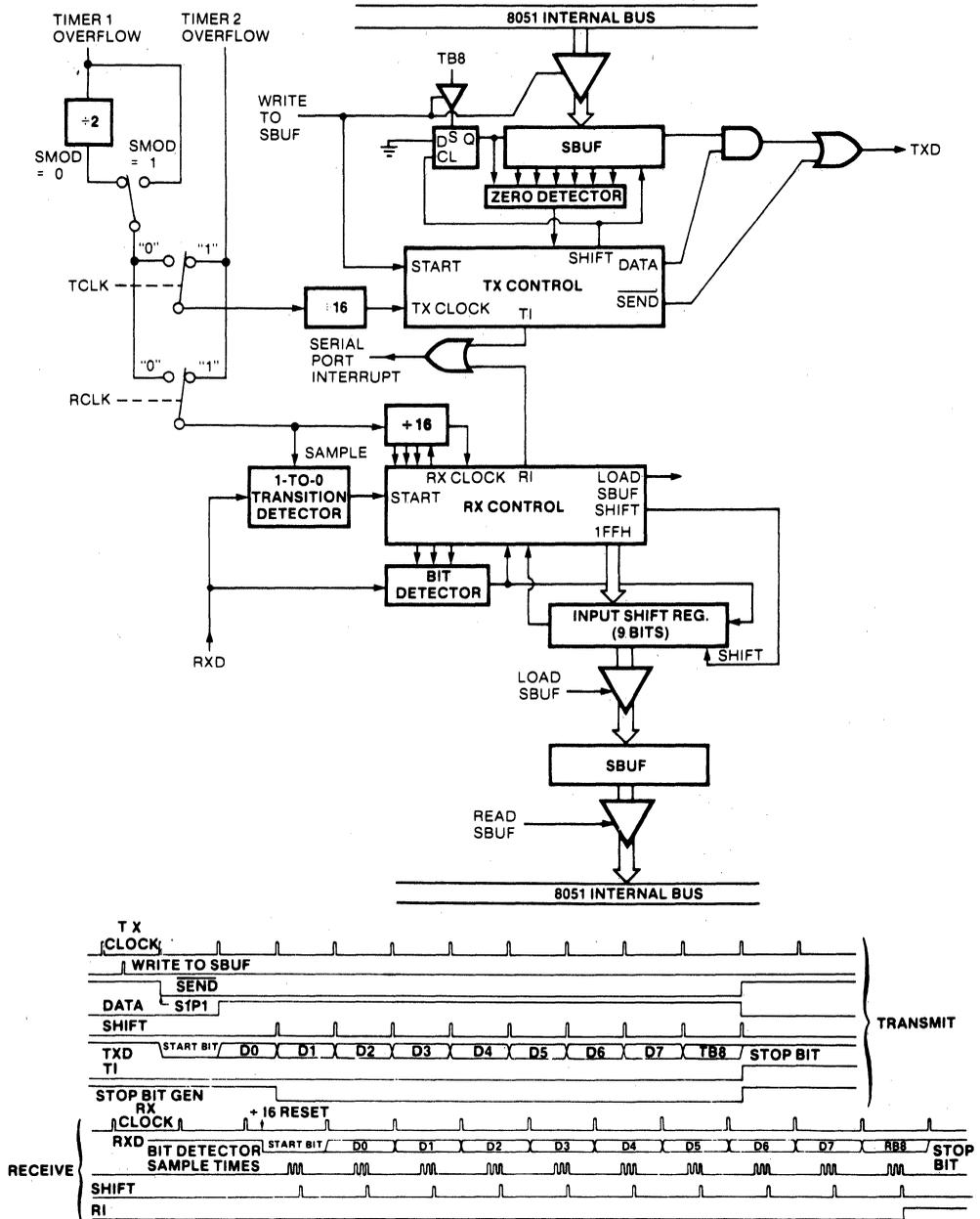
Figure 2-21b. Serial Port Mode 3

(TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.)

right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate $\overline{\text{SEND}}$ and set TI. This occurs at the 11th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written to the input shift register.

At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least two of the three samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

1) RI = 0, and
2) Either SM2 = 0, or the received data bit = 1

If either of these conditions is not met, the received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

INTERRUPTS

The 8051 provides five interrupt sources. The 8052 provides six. These are shown in Figure 2-22.

The External Interrupts $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.
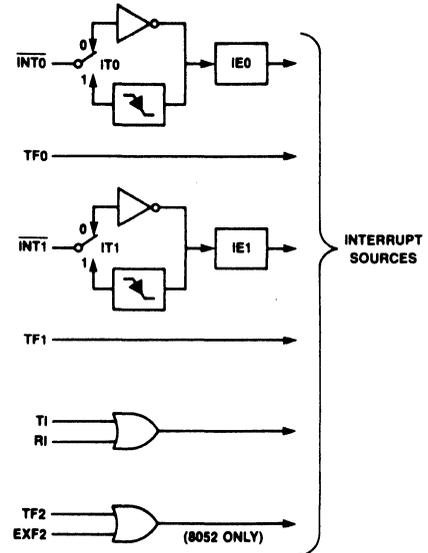
Figure 2-22. 8051 Family Interrupt Sources

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Timer 1 interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow Interrupt. If ET0 = 0, the Timer 0 Interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.

Figure 2-23. IE: Interrupt Enable Register

The Timer 0 and Timer 1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective timer/counter registers (except see page 2-12 for Timer 0 in mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In the 8052, the Timer 2 Interrupt is generated by the logical OR of TF2 and EXF2. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Figure 2-23). Note that IE contains also a global disable bit, EA, which disables all interrupts at once.

Priority Level Structure

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Figure 2-24). A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

(MSB) | | | | | | | (LSB)

| X | X | PT2 | PS | PT1 | PX1 | PT0 | PX0 |
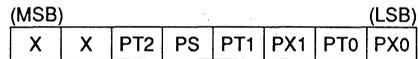
Symbol	Position	Function
—	IP.7	reserved
—	IP.6	reserved
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT = 1 programs it to the higher priority level.
PX1	IP.2	defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

Figure 2-24. IP: Interrupt Priority Register

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the **same** priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

	Source	Priority Within Level
1.	IE0	(highest)
2.	TF0	
3.	IE1	
4.	TF1	
5.	RI + TI	
6.	TF2 + EXF2	(lowest)

Note that the "priority within level" structure is only used to resolve *multiple requests of the same priority level.*
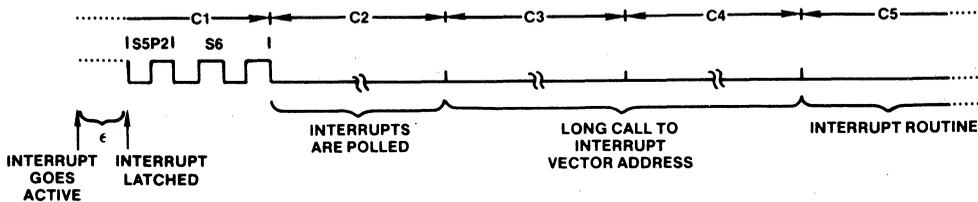
This is the fastest possible response when C2 is the final cycle of an instruction other than RETI or an access to IE or IP.

Figure 2-25. Interrupt Response Timing Diagram

How Interrupts Are Handled

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.

2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.

3. The instruction in progress is RETI or any access to the IE or IP registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least *one more* instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at S5P2 of the previous machine cycle. Note then that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not *still* active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in Figure 2-25.

Note that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in Figure 2-25, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. It never clears the Serial Port or Timer 2 flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below.

Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI + TI	0023H
TF2 + EXF2	002BH

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the $\overline{\text{INTx}}$ pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the $\overline{\text{INTx}}$ pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

Response Time

The $\overline{INT0}$ and $\overline{INT1}$ levels are inverted and latched into IE0 and IE1 at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 2-25 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 9 cycles.

SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least one instruction of the interrupted program is executed. One way to use this feature for single-step operation is to program one of the external interrupts, e.g., INT0, to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB   P3.2,$   ;WAIT HERE UNTIL INT0 GOES
               HIGH
JB    P3.2,$   ;NOW WAIT HERE UNTIL IT GOES
               LOW
RETI           ;GO BACK AND EXECUTE ONE
               INSTRUCTION
```

If the $\overline{INT0}$ pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until $\overline{INT0}$ is pulsed (from low to high to low). Then it will execute RETI, go back to the task

program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

RESET

The reset input is the RST pin, which is the input to a Schmitt Trigger.

A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods), *while the oscillator is running*. The CPU responds by executing an internal reset. It also configures the ALE and $\overline{PSEN}$ pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

Register	Content
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0–P3	0FFH
IP (8051)	XXX00000B
IP (8052)	XX000000B
IE (8051)	0XX00000B
IE (8052)	0X000000B
TMOD	00H
TCON	00H
T2CON (8052 only)	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2	00H
TL2	00H
RCAP2H (8052 only)	00H
RCAP2L (8052 only)	00H
SCON	00H
SBUF	Indeterminate
PCON (NMOS)	0XXXXXXXB
PCON (CMOS)	0XXX0000B

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless the part is returning from a reduced power mode of operation.

Power-On Reset

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10μF capacitor and to VSS through an 8.2 k resistor, providing the VCC rise time does not exceed a millisecond and the

oscillator start-up time does not exceed 10 ms. This power-on reset circuit is shown in Figure 2-26. When power comes on, the current drawn by RST commences to charge the capacitor. The voltage at RST is the difference between VCC and the capacitor voltage, and decreases from VCC as the capacitor charges. The larger the capacitor, the more slowly VRST decreases. VRST must remain above the lower threshold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

POWER-SAVING MODES OF OPERATION

For applications where power consumption is critical, the NMOS and CMOS versions provide power-reduced modes of operation.

NMOS Power Reduction Mode

To save power when using the NMOS device, VCC may be reduced to zero while the on-chip RAM is saved through a backup supply connected to the RST pin. After saving relevant data in RAM, the user enables the backup power supply to the RST pin before VCC falls below its operating limit. When power returns, the backup supply must stay on long enough to accomplish a reset; it then can be removed and normal operation resumed.

CMOS Power Reduction Modes

CMOS versions have two power-reducing modes, Idle and Power Down. Backup power is supplied during these operations through VCC. Figure 2-27 shows the internal circuitry which implements these features. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 2-28 details its contents.
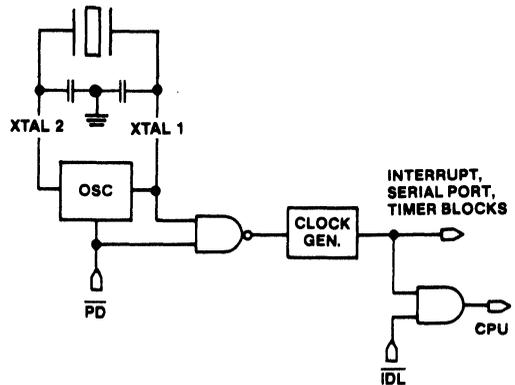
Figure 2-27. Idle and Power Down Hardware

Figure 2-26. Power on Reset Circuit

(MSB)							(LSB)
SMOD	—	—	—	GF1	GF0	PD	IDL

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1 and Timer 1 is used to generate baud rate, and the Serial Port is used in modes 1, 2, or 3.
—	PCON.6	(Reserved)
—	PCON.5	(Reserved)
—	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF2	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Idle mode bit. Setting this bit activates idle mode operation.

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).

Figure 2-28. PCON: Power Control Register

Idle Mode

An instruction that sets PCON. 0 causes that to be the last instruction executed before going into the Idle mode. In the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. ALE and $\overline{PSEN}$ hold at logic high levels.

There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into Idle.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

Power Down Mode

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped, but the on-chip RAM and Special Function Registers are held. The port pins output the values held by their respective SFRs. ALE and $\overline{PSEN}$ output lows.

The only exit from Power Down is a hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.

In the Power Down mode of operation, VCC can be reduced to minimize power consumption. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

MORE ABOUT THE ON-CHIP OSCILLATOR

NMOS Versions

The on-chip oscillator circuitry for the NMOS members of the 8051 Family is a single stage linear inverter (Figure 2-29), intended for use as a crystal-controlled, positive reactance oscillator (Figure 2-30). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.
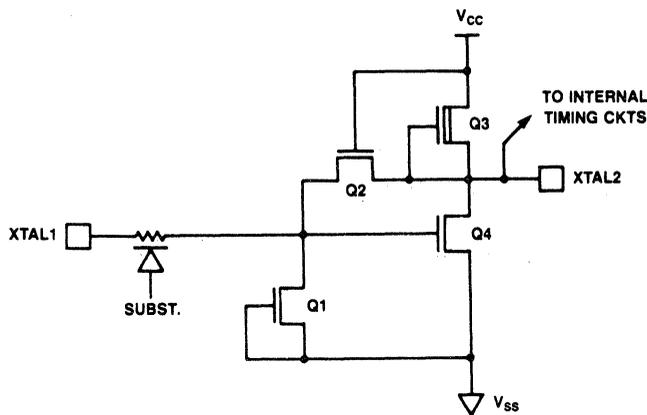
Figure 2-29. On-Chip Oscillator Circuitry in the NMOS Versions of the 8051 Family

The oscillator can be used with the same external components as the NMOS versions, as shown in Figure 2-33. Typically, C1 = C2 = 30 pF when the feedback element is a quartz crystal, and C1 = C2 = 47 pF when a ceramic resonator is used.

To drive the CMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave XTAL2 floating as shown in Figure 2-34.

The reason for this change from the way the NMOS part is driven can be seen by comparing Figure 2-29 and 2-32. In the NMOS devices the internal timing circuits are driven by the signal at XTAL2. In the CMOS devices the Internal timing circuits are driven by the signal at XTAL1.
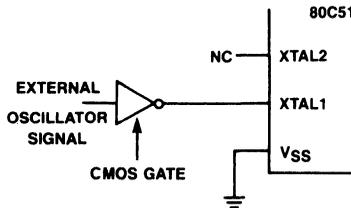
Figure 2-34. Driving the CMOS 8051 Family Parts with an External Clock Source

INTERNAL TIMING

Figures 2-35 through 2-38 show when the various strobe and port signals are clocked internally. The figures do not show rise and fall times of the signals, nor do they show propagation delays between the XTAL2 signal and events at other pins.

Rise and fall times are dependent on the external loading that each pin must drive. They are often taken to be something in the neighborhood of 10nsec, measured between 0.8 V and 2.0 V.

Propagation delays are different for different pins. For a given pin they vary with pin loading, temperature, VCC, and manufacturing lot. If the XTAL2 waveform is taken as the timing reference, propagation delays may vary from 25 to 125 nsec.

The AC Timings section of the data sheets do not reference any timing to the XTAL2 waveform. Rather, they relate the critical edges of control and input signals to each other. The timings published in the data sheets include the effects of propagation delays under the specified test conditions.

80C51BH PIN DESCRIPTIONS

VCC: Supply voltage.

VSS: Circuit ground potential.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. As an open drain output port it can sink 8 LS TTL loads. Port 0 pins that have 1s written to them float, and in that state will function as high-impedance inputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pull-ups when emitting 1s. Port 0 also emits code bytes during program verification. In that application, external pull-ups are required.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pull-ups. The port 1 output buffers can sink/source four LS TTL loads. Port 1 pins that have 1s written to them are pulled high by the internal pull-ups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pull-ups.

In the 8052, pins P1.0 and P1.1 also serve the alternate functions of T2 and T2EX. T2 is the Timer 2 external input. T2EX is the input through which a Timer 2 "capture" is triggered.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. The Port 2 output buffers can sink/source four LS TTL loads. Port 2 emits the high-order address byte during accesses to external memory that use 16-bit addresses. In this application it uses the strong internal pull-ups when emitting 1s. Port 2 also receives the high-order address and control bits during 87C51 programming and verification, and during program verification in the 80C51BH.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pull-ups. It also serves the functions of various special features of the 8051 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

The Port 3 output buffers can source/sink four LS TTL loads.

RST: Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{\text{PROG}}$: Address Latch Enable is the output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of 1/6 of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each access to external Data Memory.) This pin is also the program pulse input ($\overline{\text{PROG}}$) during EPROM programming.

$\overline{\text{PSEN}}$: Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, $\overline{\text{PSEN}}$ is activated twice each machine cycle (except that two $\overline{\text{PSEN}}$ activations are skipped during accesses to external Data Memory). $\overline{\text{PSEN}}$ is not activated when the device is executing out of internal Program Memory.

$\overline{\text{EA}}$/VPP: When $\overline{\text{EA}}$ is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH in the 80C51BH, or 1FFFH in the 80C52T2). Holding $\overline{\text{EA}}$ low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 80C31BH and 80C32T2, $\overline{\text{EA}}$ must be externally wired low. In the 87C51, this pin also receives the 12.75 V programming supply voltage (VPP) during EPROM programming.

XTAL1:   Output to the inverting oscillator amplifier (CMOS devices only).

XTAL2:   Input from the inverting oscillator amplifier (CMOS devices only).
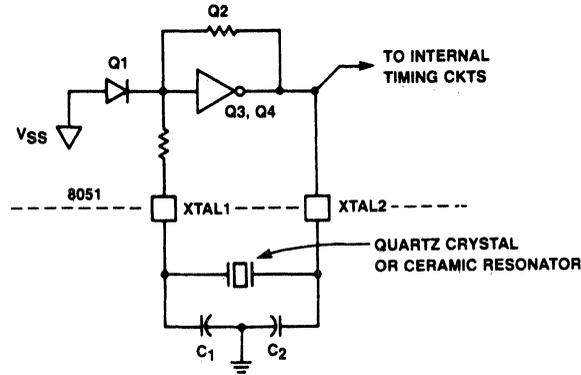
Figure 2-30. Using the NMOS On-Chip Oscillator

The crystal specifications and capacitance values (C1 and C2 in Figure 2-33) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator can be used in place of the crystal in cost-sensitive applications. When a ceramic resonator is used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

Figure 2-31. Driving the NMOS 8051 Family Parts
with an External Clock Source

To drive the NMOS parts with an external clock source, apply the external clock signal to XTAL2, and ground XTAL1, as shown in Figure 2-31. A pull-up resistor may be used (to increase noise margin), but is optional if VOH of the driving gate exceeds the VIH_{MIN} specification of XTAL2.

CMOS

The on-chip oscillator circuitry for the 80C51, shown in Figure 2-32, consists of a single-stage linear inverter intended for use as crystal-controlled, positive reactance oscillator in the same manner as the NMOS parts. However, there are some important differences.

One difference is that the 80C51 is able to turn off its oscillator under software control (by writing a 1 to the PD bit in PCON). Another difference is that in the 80C51 the internal clocking circuitry is driven by the signal at XTAL1, whereas in the NMOS versions it is by the signal at XTAL2.

The feedback resistor Rf in Figure 2-32 consists of paralleled n- and p-channel FETs controlled by the PD bit, such that Rf is opened when PD = 1. The diodes D1 and D2, which act as clamps to VCC and VSS, are parasitic to the Rf FETs.

Figure 2-32. On-Chip Oscillator Circuitry in the CMOS Versions of the 8051 Family
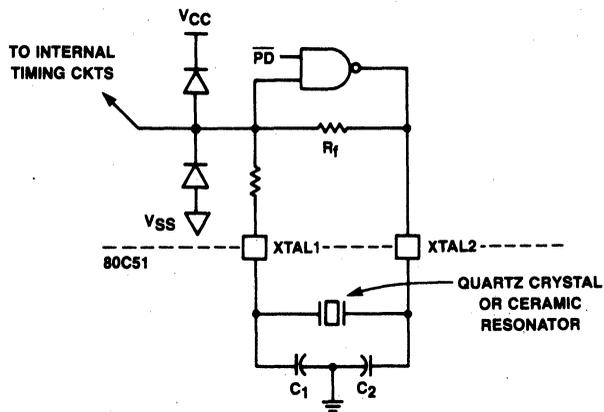
Figure 2-33. Using the CMOS On-Chip Oscillator
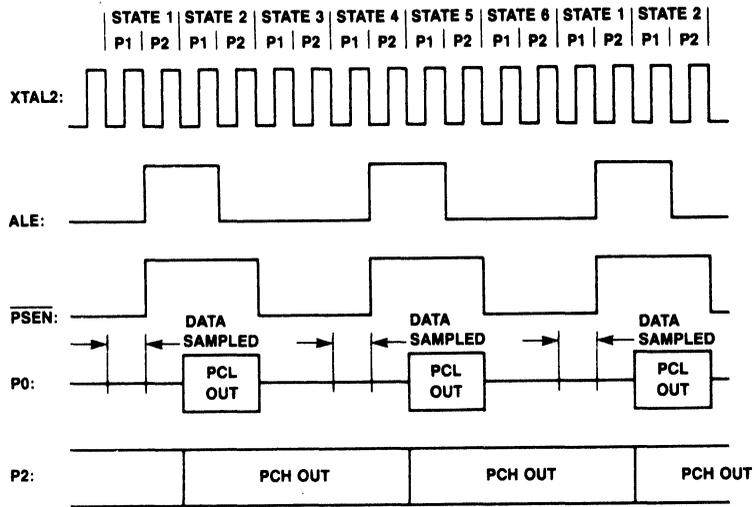
Figure 2-35. External Program Memory Fetches
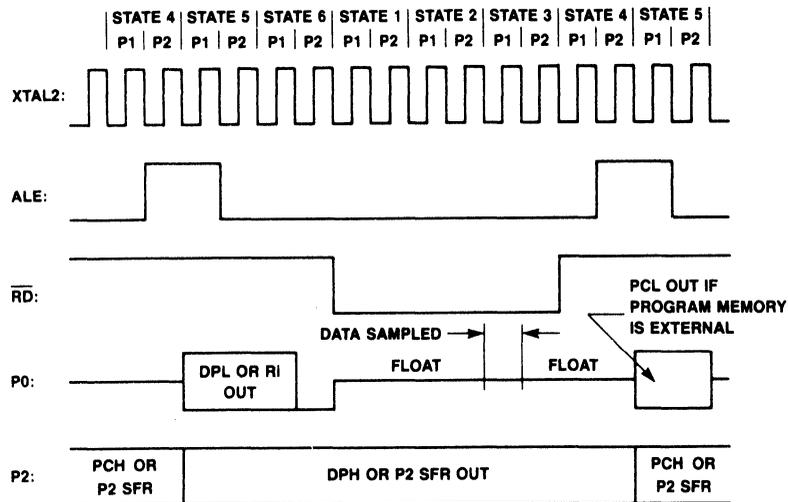
Figure 2-36. External Data Memory Read Cycle
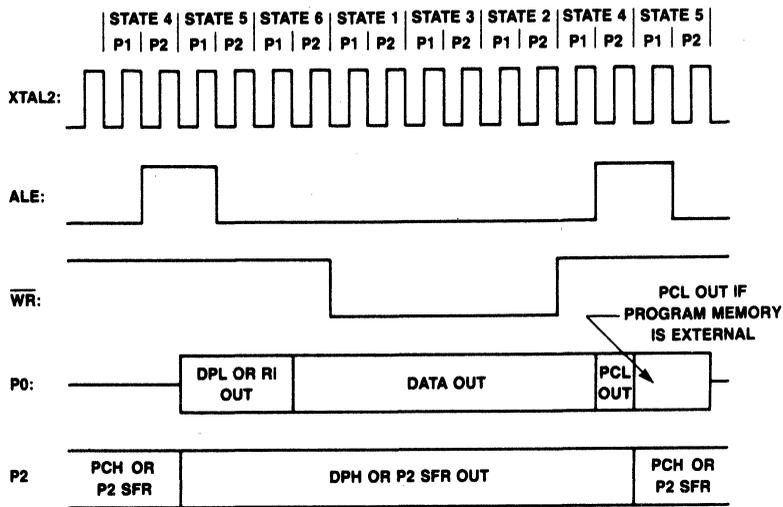
Figure 2-37. External Data Memory Write Cycle
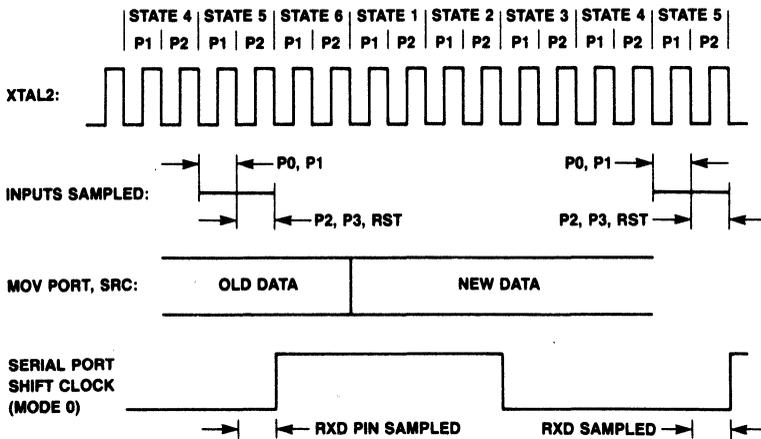
Figure 2-38. Port Operation

CHAPTER 3

Programmer's Guide

INTRODUCTION

This chapter presents a programmer's reference guide to the "core" architecture of the 8051 Family. The description of the "8051" in this chapter applies to all 8051 Family members. The term "8052" is used to refer to an 8051AH with a double amount of ROM and RAM, and an extra timer called Timer 2. It is also included in this "core" discussion because its features are often found in other enhanced 8051 Family members. (See Members of the Family in Chapter 1).

MEMORY ORGANIZATION

Program Memory

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for the 8052) may reside on-chip. Figure 3-1 shows a map of the 8051 program memory; Figure 3-2 shows a map of the 8052 program memory.
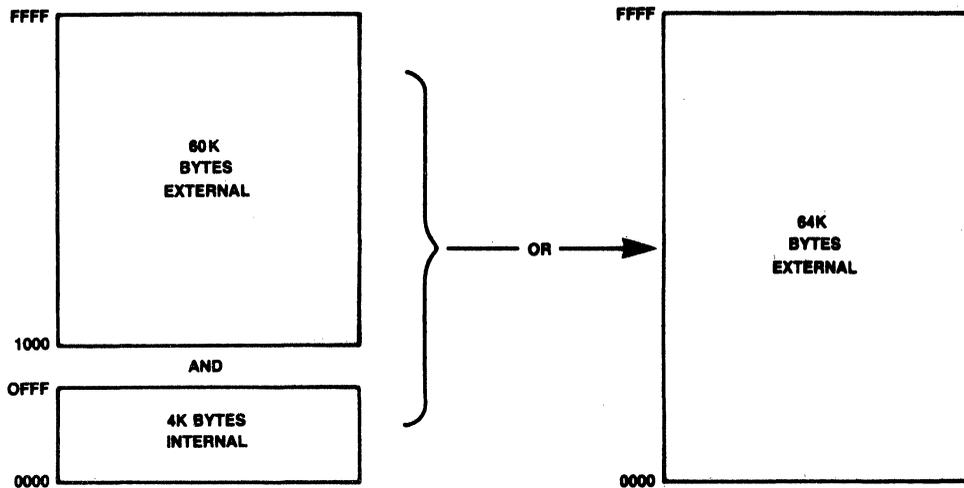
Figure 3-1. The 8051 Program Memory
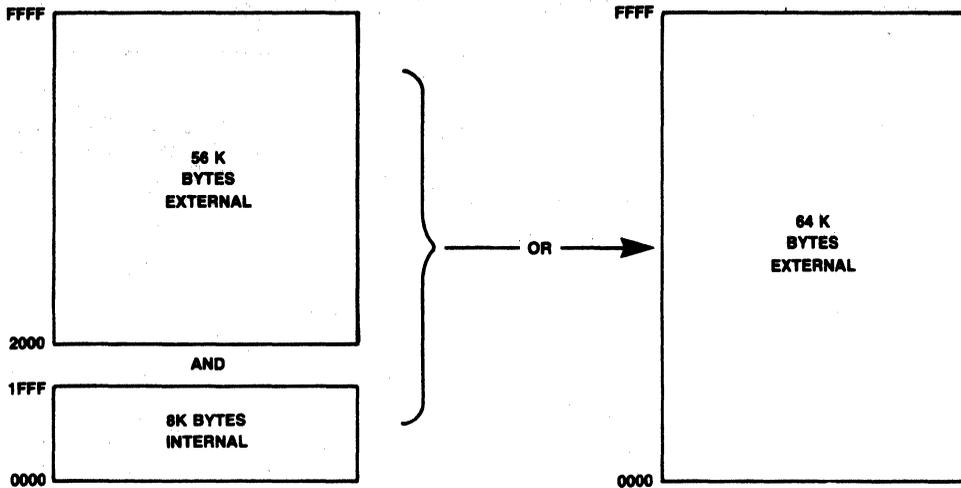
Figure 3-2. The 8052 Program Memory

Data Memory

The 8051 can address up to 64K bytes of external Data Memory. The "MOVX" instruction is used to access the external data memory. (Refer to the 8051 Family Instruction Set, in Chapter 4.)

The 8051 has 128 bytes of on-chip RAM (256 bytes in the 8052) plus a number of Special Function Registers (SFRs). The lower 128 bytes of RAM can be accessed either by direct addressing (MOV data addr) or by indirect addressing (MOV @ Ri). Figure 3-3 shows the 8051 and the 8052 Data Memory organization.
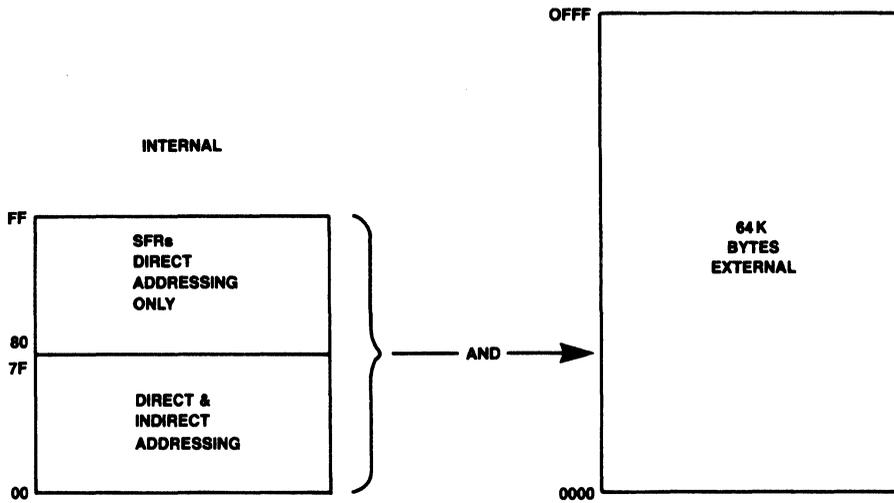
a. The 8051
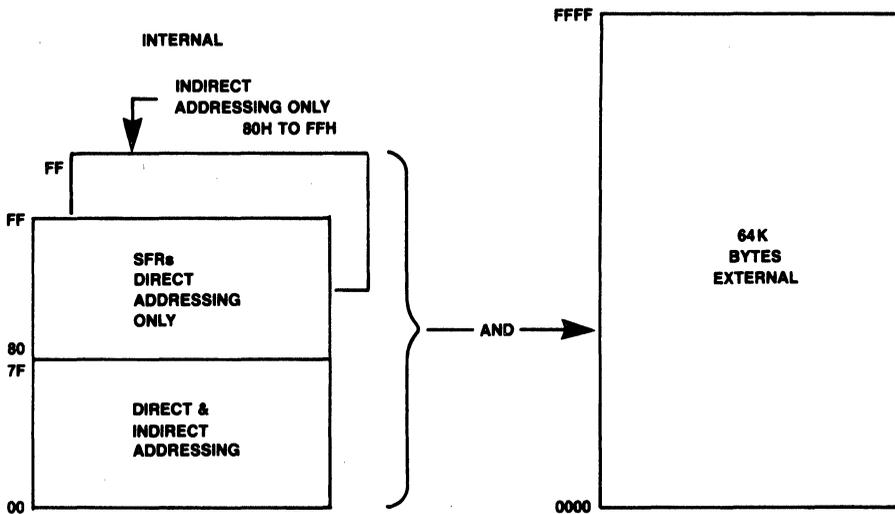
b. The 8052

Figure 3-3. Data Memory

Indirect Address Area

Figure 3-3b the SFRs and the indirect address RAM have the same addresses (80H–0FFH). Nevertheless, they are two separate areas and are accessed in two different ways.

For example, the instruction

        MOV   80H, #0AAH

writes OAAH to Port 0, which is one of the SFRs, and the instruction

        MOV   R0, #80H

        MOV   @R0, #0BBH

writes 0BBH in location 80H of the data RAM. Thus, after execution of both of the above instructions Port 0 will contain 0AAH and location 80 of the RAM will contain 0BBH.

Direct and Indirect Address Area

The 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into three segments as listed below and shown in Figure 3-4.

1. Register Banks 0-3: Locations 0 through 1FH (32 bytes). ASM-51 and the device after reset default to register bank 0. To use the other register banks the user must select them in the software. Each register bank contains eight 1-byte registers, 0 through 7.

Reset initializes the Stack Pointer to location 07H and it is incremented once to start from location 08H which is the first register (RO) of the second register bank. Thus, in order to use more than one register bank, the SP should be intialized to a different location of the RAM where it is not used for data storage (ie, higher part of the RAM).

2. Bit Addressable Area: 16 bytes have been assigned for this segment, 20H-2FH. Each one of the 128 bits of this segment can be directly addressed (0-7FH).

The bits can be referred to in two ways both of which are acceptable by the ASM-51. One way is to refer to their addresses, ie. 0 to 7FH. The other way is with reference to bytes 20H to 2FH. Thus, bits 0–7 can also be referred to as bits 20.0–20.7, and bits 8-FH are the same as 21.0–21.7 and so on.

Each of the 16 bytes in this segment can also be addressed as a byte.

3. Scratch Pad Area: Bytes 30H through 7FH are available to the user as data RAM. However, if the stack pointer has been initialized to this area, enough number of bytes should be left aside to prevent SP data destruction.
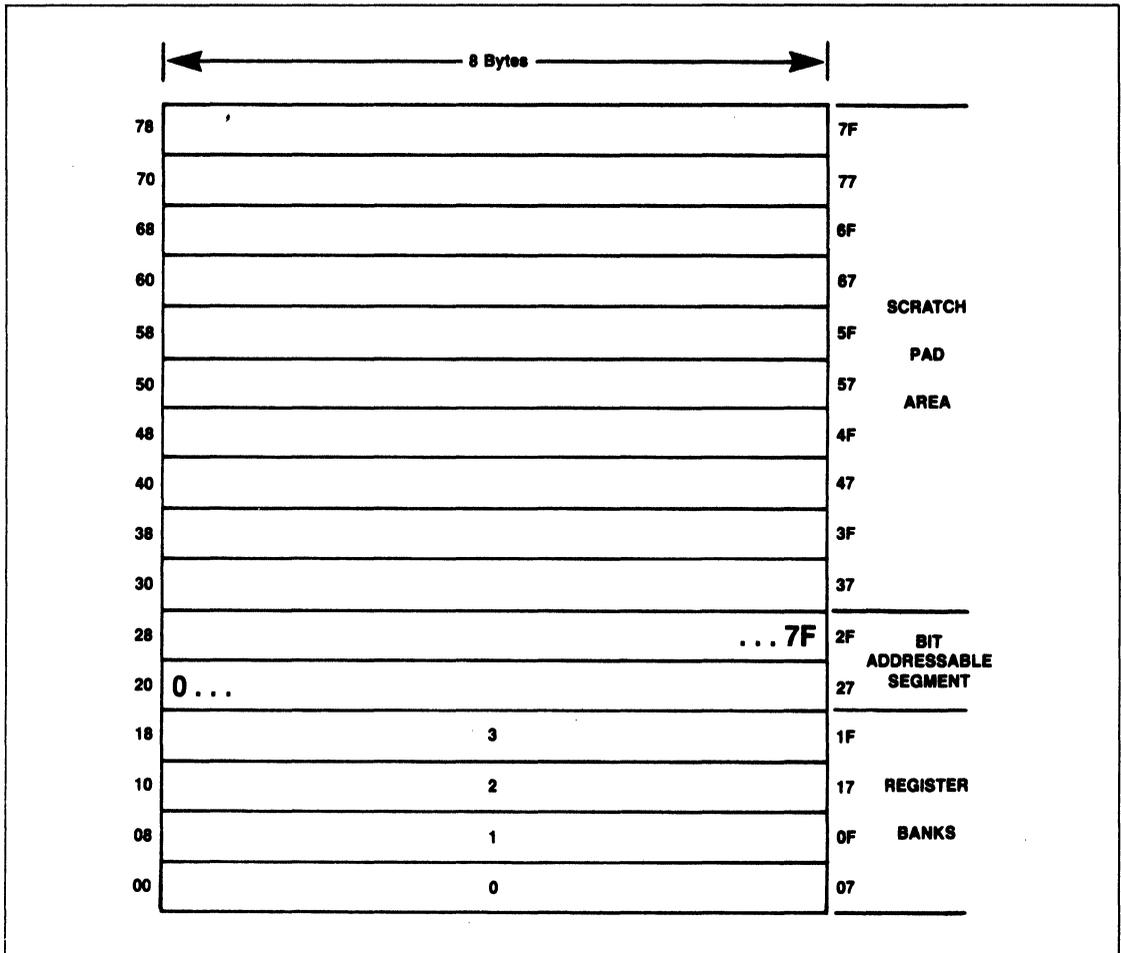
Figure 3-4. 128 Bytes of RAM Direct and Indirect Addressable

SPECIAL FUNCTION REGISTERS

Table 3-1 contains a list of all the SFRs and their addresses.

Comparing Table 3-1 and figure 3-5 shows that all of the SFRs that are byte-and bit-addressable are located on the first column in Figure 3-5.

Table 3-1

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* = Bit addressable
+ = 8052 only

What Do the SFRs Contain Just After Power-on or a Reset?

Table 3-2 lists the contents of each SFR after power-on or a hardware reset.

Table 3-2. Contents of the SFRs After Reset

Register	Value in Binary
*ACC	00000000
*B	00000000
*PSW	00000000
SP	00000111
DPTR	
DPH	00000000
DPL	00000000
*P0	11111111
*P1	11111111
*P2	11111111
*P3	11111111
*IP	8051 XXX00000,
	8052 XX000000
*IE	8051 0XX00000,
	8052 0X000000
TMOD	00000000
*TCON	00000000
*+T2CON	00000000
TH0	00000000
TL0	00000000
TH1	00000000
TL1	00000000
+TH2	00000000
+TL2	00000000
+RCAP2H	00000000
+RCAP2L	00000000
*SCON	00000000
SBUF	Indeterminate
PCON	NMOS 0XXXXXXX
	CMOS 0XXX0000

X = Undefined
*   = Bit Addressable
+ = 8052 only

SFR Memory Map

8 Bytes

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

↑
Bit
Addressable

Figure 3-5. Memory Map

Those SFRs that have their bits assigned for various functions are listed in this section. A brief description of each bit is provided for quick reference. For more detailed information refer to Architecture, Chapter 2.

PSW: Program Status Word. Bit Addressable.

CY	AC	F0	RS1	RS0	OV	—	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	Not implemented, reserved for future use.*
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

*User software should not write 1s to reserved bits. These bits may be used in future 8051 Family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

PCON: Power Control Register. Not Bit Addressable.

SMOD	—	—	—	GF1	GF0	PD	IDL

SMOD Double baud rate bit. If Timer 1 is used to generate baud rate and SMOD = 1, the baud rate is doubled when the Serial Port is used in modes 1, 2, or 3.

— Not implemented, reserved for future use.*

— Not implemented, reserved for future use.*

— Not implemented, reserved for future use.*

GF1 General purpose flag bit.

GF0 General purpose flag bit.

PD Power Down bit. Setting this bit activates Power Down operation in the 80C51BH. (Available only in CMOS).

IDL Idle Mode bit. Setting this bit activates Idle Mode operation in the 80C51BH. (Available only in CMOS).

If 1s are written to PD and IDL at the same time, PD takes precedence.

*User software should not write 1s to reserved bits. These bits may be used in future 8051 Family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

Interrupts

In order to use any of the interrupts in the 8051 Family, the following three steps must be taken.

1. Set the EA (enable all) bit in the IE register to 1.
2. Set the corresponding individual interrupt enable bit in the IE register to 1.
3. Begin the interrupt service routine at the corresponding Vector Address of that interrupt. See Table below.

Interrupt Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI & TI	0023H
TF2 & EXF2	002BH

In addition, for external interrupts, pins $\overline{INT0}$ and $\overline{INT1}$ (P3.2 and P3.3) must be set to 1, and depending on whether the interrupt is to be level or transition activated, bits IT0 or IT1 in the TCON register may need to be set to 1.

ITx = 0 level activated

ITx = 1 transition activated

IE: Interrupt Enable Register. Bit Addressable.

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	—	ET2	ES	ET1	EX1	ET0	EX0

EA   IE.7   Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

—   IE.6   Not implemented, reserved for future use.*

ET2   IE.5   Enable or disable the Timer 2 overflow or capture interrupt (8052 only).

ES   IE.4   Enable or disable the serial port interrupt.

ET1   IE.3   Enable or disable the Timer 1 overflow interrupt.

EX1   IE.2   Enable or disable External Interrupt 1.

ET0   IE.1   Enable or disable the Timer 0 overflow interrupt.

EX0   IE.0   Enable or disable External Interrupt 0.

*User software should not write 1s to reserved bits. These bits may be used in future 8051 Family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

Assigning Higher Priority to One or More Interrupts

In order to assign higher priority to an interrupt the corresponding bit in the IP register must be set to 1.

Remember that while an interrupt service is in progress, it cannot be interrupted by a lower or same level interrupt.

Priority Within Level

Priority within level is only to resolve simultaneous requests of the same priority level.

From high to low, interrupt sources are listed below:

IE0
TF0
IE1
TF1
RI or TI
TF2 or EXF2

IP: Interrupt Priority Register. Bit Addressable

If the bit is 0, the corresponding interrupt has a lower priority; if the bit is 1 the corresponding interrupt has a higher priority.

—	—	PT2	PS	PT1	PX1	PT0	PX0

—	IP. 7	Not implemented, reserved for future use.*
—	IP. 6	Not implemented, reserved for future use.*
PT2	IP. 5	Defines the Timer 2 interrupt priority level (8052 only).
PS	IP. 4	Defines the Serial Port interrupt priority level.
PT1	IP. 3	Defines the Timer 1 interrupt priority level.
PX1	IP. 2	Defines External Interrupt 1 priority level.
PT0	IP. 1	Defines the Timer 0 interrupt priority level.
PX0	IP. 0	Defines the External Interrupt 0 priority level.

*User software should not write 1s to reserved bits. These bits may be used in future 8051 Family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

TCON: Timer/Counter Control Register. Bit Addressable

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1   TCON. 7 Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.

TR1   TCON. 6 Timer 1 run control bit. Set/cleared by software to turn Timer/Counter 1 ON/OFF.

TF0   TCON. 5 Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.

TR0   TCON. 4 Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.

IE1   TCON. 3 External Interrupt 1 edge flag. Set by hardware when External Interrupt edge is detected. Cleared by hardware when interrupt is processed.

IT1   TCON. 2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

IE0   TCON. 1 External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.

IT0   TCON. 0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TMOD: Timer/Counter Mode Control Register. Not Bit Addressable

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0

          TIMER 1   TIMER 0

GATE   When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/$\overline{T}$ Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

M1   Mode selector bit. (NOTE 1)

M0   Mode selector bit. (NOTE 1)

NOTE 1:

M1	M0		Operating Mode
0	0	0	13-bit Timer (8048 Family compatible)
0	1	1	16-bit Timer/Counter
1	0	2	8-bit Auto-Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8-bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

TIMER SET-UP

Tables 3-3 through 3-6 give some values for TMOD which can be used to set up Timer 0 in different modes.

It is assumed that only one timer is being used at a time. If it is desired to run Timer 0 and 1 simultaneously, in any mode, the value in TMOD for Timer 0 must be ORed with the value shown for Timer 1 (Tables 3-5 and 3-6).

For example, if it is desired to run Timer 0 in mode 1 GATE (external control), and Timer 1 in mode 2 COUNTER, then the value that must be loaded into TMOD is 69H (09H from Table 3-3 Ored with 60H from Table 3-6).

Moreover, it is assumed that the user, at this point, is not ready to turn the timers on and will do that at a different point in the program by setting bit TRx (in TCON) to 1.

Timer/Counter 0

As a Timer:

Table 3-3

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	08H
1	16-bit Timer	01H	09H
2	8-bit Auto-Reload	02H	0AH
3	two 8-bit Timers	03H	0BH

As a Counter:

Table 3-4

MODE	COUNTER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	04H	0CH
1	16-bit Timer	05H	0DH
2	8-bit Auto-Reload	06H	0EH
3	one 8-bit Counter	07H	0FH

NOTES:
1. The Timer is turned ON/OFF by setting/clearing bit TR0 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on $\overline{INT0}$ (P3.2) when TR0 = 1 (hardware control).

Timer/Counter 1

As a Timer:

Table 3-5

MODE	TIMER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	80H
1	16-bit Timer	10H	90H
2	8-bit Auto-Reload	20H	A0H
3	does not run	30H	B0H

As a Counter:

Table 3-6

MODE	COUNTER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	40H	C0H
1	16-bit Timer	50H	D0H
2	8-bit Auto-Reload	60H	E0H
3	not available	—	—

NOTES:
1. The Timer is turned ON/OFF by setting/clearing bit TR1 in the software.
2. The timer is turned ON/OFF by the 1-to-0 transition on $\overline{INT1}$ (P3.3) when TR1 = 1 (hardware control).

T2CON: TIMER/COUNTER 2 CONTROL REGISTER. BIT ADDRESSABLE.

8052 Only

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

TF2   T2CON. 7   Timer 2 overflow flag set by hardware and cleared by software. TF2 cannot be set when either RCLK = 1 or CLK = 1

EXF2   T2CON. 6   Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX, and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.

RCLK   T2CON. 5   Receive clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its receive clock in modes 1 & 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.

TLCK   T2CON. 4   Transmit clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its transmit clock in modes 1 & 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.

EXEN2   T2CON. 3   Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of negative transition on T2EX if Timer 2 is not being used to clock the Serial Port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.

TR2   T2CON. 2   Software START/STOP control for Timer 2. A logic 1 starts the Timer.

C/$\overline{T2}$ T2CON. 1   Timer or Counter select.

             0 = Internal Timer. 1 = External Event Counter (falling edge triggered).

CP/$\overline{RL2}$ T2CON. 0   Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, Auto-Reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the Timer is forced to Auto-Reload on Timer 2 overflow.

Timer/Counter 2 Set-up

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set separately to turn the Timer on.

As a Timer:

Table 3-7

MODE	T2CON	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16-bit Auto-Reload	00H	08H
16-bit Capture	01H	09H
BAUD rate generator receive & transmit same baud rate	34H	36H
receive only	24H	26H
transmit only	14H	16H

As a Counter:

Table 3-8

MODE	TMOD	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16-bit Auto-Reload	02H	0AH
16-bit Capture	03H	0BH

NOTES:
1. Capture/Reload occurs only on Timer/Counter overflow.
2. Capture/Reload occurs on Timer/Counter overflow and a 1 to 0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baud rate generating mode.

SCON: SERIAL PORT CONTROL REGISTER. BIT ADDRESSABLE.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 SCON. 7 Serial Port mode specifier. (NOTE 1).

SM1 SCON. 6 Serial Port mode specifier. (NOTE 1).

SM2 SCON. 5 Enables the multiprocessor communication feature in modes 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. (See Table 9).

REN SCON. 4 Set/Cleared by software to Enable/Disable reception.

TB8 SCON. 3 The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.

RB8 SCON. 2 In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.

TI SCON. 1 Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.

RI SCON. 0 Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8-Bit UART	Variable
1	0	2	9-Bit UART	Fosc./64 OR Fosc./32
1	1	3	9-Bit UART	Variable

Serial Port Set-up

Table 3-9

MODE	SCON	SM2 VARIATION
0	10H	Single Processor Environment (SM2 = 0)
1	50H	
2	90H	
3	D0H	
0	NA	Multiprocessor Environment (SM2 = 1)
1	70H	
2	B0H	
3	F0H	

GENERATING BAUD RATES

Serial Port in Mode 0

Mode 0 has a fixed baud rate which is 1/12 of the oscillator frequency. To run the serial port in this mode none of the Timer/Counters need to be set up. Only the SCON register needs to be defined.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{12}$$

Serial Port in Mode 1

Mode 1 has a variable baud rate. The baud rate can be generated by either Timer 1 or Timer 2 (8052 only).

Using Timer/Counter 1 to Generate Baud Rates:

For this purpose, Timer 1 is used in mode 2 (Auto-Reload). Refer to Timer Setup section of this chapter.

$$\text{Baud Rate} = \frac{K \times \text{Oscillator Freq.}}{32 \times 12 \times [256 - (TH1)]}$$

If SMOD = 0, then K = 1.
If SMOD = 1, then K = 2. (SMOD is the PCON register).

Most of the time the user knows the baud rate and needs to know the reload value for TH1.
Therefore, the equation to calculate TH1 can be written as:

$$TH1 = 256 - \frac{K \times \text{Osc Freq.}}{384 \times \text{baud rate}}$$

TH1 must be an integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.

Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register. (ie, ORL PCON, #80H). The address of PCON is 87H.

Using Timer/Counter 2 to Generate Baud Rates:

For this purpose, Timer 2 must be used in the baud rate generating mode. Refer to Timer 2 Setup Table in this chapter. If Timer 2 is being clocked through pin T2 (P1.0) the baud rate is:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

And if it is being clocked internally the baud rate is:

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times [65536 - (RCAP2H, RCAP2L)]}$$

To obtain the reload value for RCAP2H and RCAP2L, the above equation can be rewritten as:

$$RCAP2H, RCAP2L = 65536 - \frac{\text{Osc Freq}}{32 \times \text{Baud Rate}}$$

Serial Port in Mode 2

The baud rate is fixed in this mode and is $\frac{1}{32}$ or $\frac{1}{64}$ of the oscillator frequency depending on the value of the SMOD bit in the PCON register.

In this mode none of the Timers are used and the clock comes from the internal phase 2 clock.

SMOD = 1, Baud Rate = $\frac{1}{32}$ Osc Freq.

SMOD = 0, Baud Rate = $\frac{1}{64}$ Osc Freq.

To set the SMOD bit: ORL PCON, #80H. The address of PCON is 87H.

Serial Port in Mode 3

The baud rate in mode 3 is variable and sets up exactly the same as in mode 1.

CHAPTER 4

Instruction Set

INTRODUCTION

All members of the 8051 Family execute the same instruction set, optimized for 8-bit control applications. The instruction set provides a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. It provides extensive support for one-bit variables as a separate data type, allowing direct bit manipulation in control and logic systems that requrie Boolean processing. An overview of the instruction set is presented below, with a brief description of how certain instructions might be used.

PROGRAM STATUS WORD

The Program Status Word (PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown in Figure 4-1, resides in SFR space. It contains the Carry bit, the Auxiliary Carry (for BCD operations), the two register bank select bits, the Overflow flag, a Parity bit, and two user-definable status flags.

The Carry bit, other than serving the functions of a Carry bit in arithmetic operations, also serves as the "Accumulator" for a number of Boolean operations.

The bits RS0 and RS1 are used to select one of the four register banks shown in Figure 1-7 . A number of instructions refer to these RAM locations as R0 through R7. The selection of which of the four banks is being referred to is made on the basis of the bits RS0 and RS1 at execution time.

The Parity bit reflects the number of 1s in the Accumulator: P = 1 if the Accumulator contains an odd number of 1s, and P = 0 if the Accumulator contains an even number of 1s. Thus the number of 1s in the Accumulator plus P is always even.

Two bits in the PSW are uncommitted and may be used as general purpose status flags.

ADDRESSING MODES

The addressing modes in the 8051 Family instruction set are as follows:

Direct Addressing

In direct addressing the operand is specified by an 8-bit address field in the instruction. Only internal Data RAM and SFRs can be directly addressed.

Indirect Addressing

In indirect addressing the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed.

The address register for 8-bit addresses can be R0 or R1 of the selected register bank, or the Stack Pointer. The address register for 16-bit addresses can only be the 16-bit "data pointer" register, DPTR.

CY	AC	F0	RS1	RS0	OV		P

PSW 7 —
CARRY FLAG RECEIVES CARRY OUT
FROM BIT 1 OF ALU OPERANDS

PSW 0
PARITY OF ACCUMULATOR SET
BY HARDWARE TO 1 IF IT CONTAINS
AN ODD NUMBER OF 1S, OTHERWISE
IT IS RESET TO 0

PSW 6 —
AUXILIARY CARRY FLAG RECEIVES
CARRY OUT FROM BIT 1 OF
ADDITION OPERANDS

PSW 1
USER DEFINABLE FLAG

PSW 5 —
GENERAL PURPOSE STATUS FLAG

PSW 2
OVERFLOW FLAG SET BY
ARITHMETIC OPERATIONS

PSW 4 —
REGISTER BANK SELECT BIT 1

PSW 3
REGISTER BANK SELECT BIT 0

Figure 4-1. PSW (Program Status Word) Register in 8051 Family Devices

Register Instructions

The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code efficient, since this mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank is accessed. One of four banks is selected at execution time by the two bank select bits in the PSW.

Register-Specific Instructions

Some instructions are specific to a certain register. For example, some instructions always operate on the Accumulator, or Data Pointer, etc., so no address byte is needed to point to it. The opcode itself does that. Instructions that refer to the Accumlator as A assemble as accumulator-specific opcodes.

Immediate Constants

The value of a constant can follow the opcode in Program Memory. For example,

MOV A, #100

loads the Accumulator with the decimal number 100. The same number could be specified in hex digits as 64H.

Indexed Addressing

Only Program Memory can be accessed with indexed addressing, and it can only be read. This addressing mode is intended for reading look-up tables in Program Memory. A 16-bit base register (either DPTR or the Program Counter) points to the base of the table, and the Accumulator is set up with the table entry number. The address of the table entry in Program Memory is formed by adding the Accumulator data to the base pointer.

Another type of indexed addressing is used in the "case jump" instruction. In this case the destination address of a jump instruction is computed as the sum of the base pointer and the Accumulator data.

ARITHMETIC INSTRUCTIONS

The menu of arithmetic instructions is listed in Table 4-1. The table indicates the addressing modes that can be used with each instruction to access the <byte> operand. For example, the ADD A, <byte> instruction can be written as:

ADD   A,7FH   (direct addressing)
ADD   A,@R0   (indirect addressing)
ADD   A,R7   (register addressing)
ADD   A, #127   (immediate constant)

Table 4-1. A List of the 8051 Family Arithmetic Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ADD A,<byte>	A = A + <byte>	X	X	X	X	1
ADDC A,<byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A,<byte>	A = A − <byte> − C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A − 1	Accumulator only				1
DEC <byte>	<byte> = <byte> − 1	X	X	X		1
MUL AB	B:A = B x A	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

The execution times listed in Table 4-1 assume a 12MHz clock frequency. All of the arithmetic instructions execute in 1 μs except the INC DPTR instruction, which takes 2 μs, and the Multiply and Divide instructions, which take 4 μs.

Note that any byte in the internal Data Memory space can be incremented or decremented without going through the Accumulator.

One of the INC instructions operates on the 16-bit Data Pointer. The Data Pointer is used to generate 16-bit addresses for external memory, so being able to increment it in one 16-bit operation is a useful feature.

The MUL AB instruction multiplies the Accumulator by the data in the B register and puts the 16-bit product into the concatenated B and Accumulator registers.

The DIV AB instruction divides the Accumulator by the data in the B register and leaves the 8-bit quotient in the Accumulator, and the 8-bit remainder in the B register.

Oddly enough, DIV AB finds less use in arithmetic "divide" routines than in radix conversions and programmable shift operations. An example of the use of DIV AB in a radix conversion will be given later. In shift operations, dividing a number by 2^n shifts its n bits to the right. Using DIV AB to perform the division completes the shift in 4 μs and leaves the B register holding the bits that were shifted out.

The DA A instruction is for BCD arithmetic operations. In BCD arithmetic, ADD and ADDC instructions should always be followed by a DA A operation, to ensure that the result is also in BCD. Note that DA A will not convert a binary number to BCD. The DA A operation produces a meaningful result only as the second step in the addition of two BCD bytes.

LOGICAL INSTRUCTIONS

Table 4-2 shows the list of 8051 Family logical instructions. The instructions that perform Boolean operations (AND, OR, Exclusive OR, NOT) on bytes perform the operation on a bit-by-bit basis. That is, if the Accumulator contains 00110101B and <byte> contains 01010011B, then

    ANL   A, <byte>

will leave the Accumulator holding 00010001B.

Table 4-2. A List of the 8051 Family Logical Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ANL A, <byte>	A = A .AND. <byte>	X	X	X	X	1
ANL <byte>,A	<byte> = <byte> .AND. A	X				1
ANL <byte>,#data	<byte> = <byte> .AND. #data	X				2
ORL A, <byte>	A = A .OR. <byte>	X	X	X	X	1
ORL <byte>,A	<byte> = <byte> .OR. A	X				1
ORL <byte>,#data	<byte> = <byte> .OR. #data	X				2
XRL A, <byte>	A = A .XOR. <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> .XOR. A	X				1
XRL <byte>,#data	<byte> = <byte> .XOR. #data	X				2
CRL A	A = 00H	Accumulator only				1
CPL A	A = .NOT. A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

The addressing modes that can be used to access the <byte> operand are listed in Table 3. Thus, the ANL A,<byte> instruction may take any of the forms

ANL	A,7FH	(direct addressing)
ANL	A,@R1	(indirect addressing)
ANL	A,R6	(register addressing)
ANL	A,#53H	(immediate constant)

All of the logical instructions that are Accumulator-specific execute in 1μs (using a 12 MHz clock). The others take 2 μs.

Note that Boolean operations can be performed on any byte in the internal Data Memory space without going through the Accumulator. The XRL <byte>, #data instruction, for example, offers a quick and easy way to invert port bits, as in

XRL   P1,#0FFH

If the operation is in response to an interrupt, not using the Accumulator saves the time and effort to stack it in the service routine.

The Rotate instructions (RL A, RLC A, etc.) shift the Accumulator 1 bit to the left or right. For a left rotation, the MSB rolls into the LSB position. For a right rotation, the LSB rolls into the MSB position.

The SWAP A instruction interchanges the high and low nibbles within the Accumulator. This is a useful operation in BCD manipulations. For example, if the Accumulator contains a binary number which is known to be less than 100, it can be quickly converted to BCD by the following code:

```
MOV   B,#10
DIV   AB
SWAP  A
ADD   A,B
```

Dividing the number by 10 leaves the tens digit in the low nibble of the Accumulator, and the ones digit in the B register. The SWAP and ADD instructions move the tens digit to the high nibble of the Accumulator, and the ones digit to the low nibble.

DATA TRANSFERS

Internal RAM

Table 4-3 shows the menu of instructions that are available for moving data around within the internal memory spaces, and the addressing modes that can be used with each one. With a 12 MHz clock, all of these instructions execute in either 1 or 2 μs.

The MOV <dest>, <src> instruction allows data to be transferred between any two internal RAM or SFR locations without going through the Accumulator. Remember the Upper 128 byes of data RAM can be accessed only by indirect addressing, and SFR space only by direct addressing.

Note that in all 8051 Family devices, the stack resides in on-chip RAM, and grows upwards. The PUSH instruction first increments the Stack Pointer (SP), then copies the byte into the stack. PUSH and POP use only direct addressing to identify the byte being saved or restored, but the stack itself is accessed by indirect addressing using the SP register. This means the stack can go into the Upper 128, if they are implemented, but not into SFR space.

Table 4-3. 8051 Family Data Transfer Instructions that Access Internal Data Memory Space

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
MOV A,<src>	A = <src>	X	X	X	X	1
MOV <dest>,A	<dest> = A	X	X	X		1
MOV <dest>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	DPTR = 16-bit immediate constant.				X	2
PUSH <src>	INC SP : MOV "@SP",<src>	X				2
POP <dest>	MOV <dest>, "@SP" : DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles		X			1

		2A	2B	2C	2D	2E	ACC
MOV	A,2EH	00	12	34	56	78	78
MOV	2EH,2DH	00	12	34	56	56	78
MOV	2DH,2CH	00	12	34	34	56	78
MOV	2CH,2BH	00	12	12	34	56	78
MOV	2BH,#0	00	00	12	34	56	78

(a) Using direct MOVs: 14 bytes, 9 μs

		2A	2B	2C	2D	2E	ACC
CLR	A	00	12	34	56	78	00
XCH	A,2BH	00	00	34	56	78	12
XCH	A,2CH	00	00	12	56	78	34
XCH	A,2DH	00	00	12	34	78	56
XCH	A,2EH	00	00	12	34	56	78

(b) Using XCHs: 9 bytes, 5 μs

**Figure 4-2. Shifting a BCD Number
Two Digits to the Right**

		2A	2B	2C	2D	2E	ACC
MOV	R1,#2EH	00	12	34	56	78	XX
MOV	R0,#2DH	00	12	34	56	78	XX
loop for R1 = 2EH:							
LOOP: MOV	A,@R1	00	12	34	56	78	78
XCHD	A,@R0	00	12	34	58	78	76
SWAP	A	00	12	34	58	78	67
MOV	@R1,A	00	12	34	58	67	67
DEC	R1	00	12	34	58	67	67
DEC	R0	00	12	34	58	67	67
CJNE	R1,#2AH,LOOP						
loop for R1 = 2DH:		00	12	38	45	67	45
loop for R1 = 2CH:		00	18	23	45	67	23
loop for R1 = 2BH:		08	01	23	45	67	01
CLR	A	08	01	23	45	67	00
XCH	A,2AH	00	01	23	45	67	08

**Figure 4-3. Shifting a BCD Number
One Digit to the Right**

The Upper 128 are not implemented in 8051 Family devices with 128 bytes of RAM. With these devices, if the SP points to the Upper 128, PUSHed bytes are lost, and POPed bytes are indeterminate.

The Data Transfer instructions include a 16-bit MOV that can be used to initialize the Data Pointer (DPTR) for look-up tables in Program Memory, or for 16-bit external Data Memory accesses.

The XCH A, <byte> instruction causes the Accumulator and addressed byte to exchange data. The XCHD A,@Ri instruction is similar, but only the low nibbles are involved in the exchange.

To see how XCH and XCHD can be used to facilitate data manipulations, consider first the problem of shifting an 8-digit BCD number two digits to the right. Figure 4-2 shows how this can be done using direct MOVs, and for comparison how it can be done using XCH instructions. To aid in understanding how the code works, the contents of the registers that are holding the BCD number and the content of the Accumulator are shown alongside each instruction to indicate their status after the instruction has been executed.

After the routine has been executed, the Accumulator contains the two digits that were shifted out on the right. Doing the routine with direct MOVs uses 14 code bytes and 9 μs of execution time (assuming a 12 MHz clock). The same operation with XCHs uses less code and executes almost twice as fast.

To right-shift by an odd number of digits, a one-digit shift must be executed. Figure 4-3 shows a sample of code that will right-shift a BCD number one digit, using the XCHD instruction. Again, the contents of the registers holding the number and of the Accumulator are shown alongside each instruction.

First, pointers R1 and R0 are set up to point to the two bytes containing the last four BCD digits. Then a loop is executed which leaves the last byte, location 2EH, holding the last two digits of the shifted number. The pointers are decremented, and the loop is repeated for location 2DH. The CJNE instruction (Compare and Jump if Not Equal) is a loop control that will be described later.

The loop is executed from LOOP to CJNE for R1 = 2EH, 2DH, 2CH and 2BH. At that point the digit that was originally shifted out on the right has propagated to location 2AH. Since that location should be left with 0s, the lost digit is moved to the Accumulator.

External RAM

Table 4-4 shows a list of the Data Transfer instructions that access external Data Memory. Only indirect addressing can be used. The choice is whether to use a one-byte address, @Ri, where Ri can be either R0 or

R1 of the selected register bank, or a two-byte address, @DPTR. The disadvantage to using 16-bit addresses if only a few K bytes of external RAM are involved is that 16-bit addresses use all 8 bits of Port 2 as address bus. On the other hand, 8-bit addresses allow one to address a few K bytes of RAM, as shown in Figure 1-5, without having to sacrifice all of Port 2.

All of these instructions execute in 2 µs, with a 12 MHz clock.

Table 4-4. 8051 Family Data Transfer
Instructions that Access
External Data Memory Space

Address Width	Mnemonic	Operation	Execution Time (µs)
8 bits	MOVX A,@Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri,A	Write external RAM @Ri	2
16 bits	MOVX A,@DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @DPTR	2

Note that in all external Data RAM accesses, the Accumulator is always either the destination or source of the data.

The read and write strobes to external RAM are activated only during the execution of a MOVX instruction. Normally these signals are inactive, and in fact if they're not going to be used at all, their pins are available as extra I/O lines. More about that later.

Lookup Tables

Table 4-5 shows the two instructions that are available for reading lookup tables in Program Memory. Since these instructions access only Program Memory, the lookup tables can only be read, not updated. The mnemonic is MOVC for "move constant".

If the table access is to external Program Memory, then the read strobe is $\overline{\text{PSEN}}$.

Table 4-5. The 8051 Family
Lookup Table Read Instructions

Mnemonic	Operation	Execution Time (µs)
MOVC A,@A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A,@A+PC	Read Pgm Memory at (A+PC)	2

The first MOVC instruction in Table 4-5 can accommodate a table of up to 256 entries, numbered 0 through 255. The number of the desired entry is loaded into the Accumulator, and the Data Pointer is set up to point to beginning of the table. Then

MOVC   A,@A+DPTR

copies the desired table entry into the Accumulator.

The other MOVC instruction works the same way, except the Program Counter (PC) is used as the table base, and the table is accessed through a subroutine. First the number of the desired entry is loaded into the Accumulator, and the subroutine is called:

MOV   A,ENTRY_NUMBER
CALL   TABLE

The subroutine "TABLE" would look like this:

TABLE: MOVC   A,@A+PC
        RET

The table itself immediately follows the RET (return) instruction in Program Memory. This type of table can have up to 255 entries, numbered 1 through 255. Number 0 can not be used, because at the time the MOVC instruction is executed, the PC contains the address of the RET instruction. An entry numbered 0 would be the RET opcode itself.

BOOLEAN INSTRUCTIONS

8051 Family devices contain a complete Boolean (single-bit) processor. The internal RAM contains 128 addressable bits, and the SFR space can support up to 128 other addressable bits. All of the port lines are bit-addressable, and each one can be treated as a separate single-bit port. The instructions that access these bits are not just conditional branches, but a complete menu of move, set, clear, complement, OR, and AND instructions. These kinds of bit operations are not easily obtained in other architectures with any amount of byte-oriented software.

Table 4-6. A List of the 8051 Family Boolean Instructions

Mnemonic		Operation	Execution Time (μs)
ANL	C,bit	C = C .AND. bit	2
ANL	C,/bit	C = C .AND. .NOT. bit	2
ORL	C,bit	C = C .OR. bit	2
ORL	C,/bit	C = C .OR. .NOT. bit	2
MOV	C,bit	C = bit	1
MOV	bit,C	bit = C	2
CLR	C	C = 0	1
CLR	bit	bit = 0	1
SETB	C	C = 1	1
SETB	bit	bit = 1	1
CPL	C	C = .NOT. C	1
CPL	bit	bit = .NOT. bit	1
JC	rel	Jump if C = 1	2
JNC	rel	Jump if C = 0	2
JB	bit,rel	Jump if bit = 1	2
JNB	bit,rel	Jump if bit = 0	2
JBC	bit,rel	Jump if bit = 1; CLR bit	2

The instruction set for the Boolean processor is shown in Table 4-6. All bit accesses are by direct addressing. Bit addresses 00H through 7FH are in the Lower 128, and bit addresses 80H through FFH are in SFR space.

Note how easily an internal flag can be moved to a port pin:

```
MOV     C,FLAG
MOV     P1.0,C
```

In this example, FLAG is the name of any addressable bit in the Lower 128 or SFR space. An I/O line (the LSB of Port 1, in this case) is set or cleared depending on whether the flag bit is 1 or 0.

The Carry bit in the PSW is used as the single-bit Accumulator of the Boolean processor. Bit instructions that refer to the Carry bit as C assemble as Carry-specific instructions (CLR C, etc). The Carry bit also has a direct address, since it resides in the PSW register, which is bit-addressable.

Note that the Boolean instruction set includes ANL and ORL operations, but not the XRL (Exclusive OR) operation. An XRL operation is simple to implement in software. Suppose, for example, it is required to form the Exclusive OR of two bits:

C = bit1 .XRL. bit2

The software to do that could be as follows:

```
        MOV     C,bit1
        JNB     bit2,OVER
        CPL     C
OVER:   (continue)
```

First, bit1 is moved to the Carry. If bit2 = 0, then C now contains the correct result. That is, bit1 .XRL. bit2 = bit1 if bit2 = 0. On the other hand, if bit2 = 1 C now contains the complement of the correct result. It need only be inverted (CPL C) to complete the operation.

This code uses the JNB instruction, one of a series of bit-test instructions which execute a jump if the addressed bit is set (JC, JB, JBC) or if the addressed bit is not set (JNC, JNB). In the above case, bit2 is being tested, and if bit2 = 0 the CPL C instruction is jumped over.

JBC executes the jump if the addressed bit is set, and also clears the bit. Thus a flag can be tested and cleared in one operation.

All the PSW bits are directly addressable, so the Parity bit, or the general purpose flags, for example, are also available to the bit-test instructions.

Relative Offset

The destination address for these jumps is specified to the assembler by a label or by an actual address in Program Memory. However, the destination address assembles to a **relative** offset byte. This is a signed (two's complement) offset byte which is added to the PC in two's complement arithmetic if the jump is executed.

The range of the jump is therefore -128 to $+127$ Program Memory bytes relative to the first byte following the instruction.

JUMP INSTRUCTIONS

Table 4-7 shows the list of unconditional jumps.

Table 4-7. Unconditional Jumps in 8051 Family Devices

Mnemonic	Operation	Execution Time (μs)
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

The Table lists a single "JMP addr" instruction, but in fact there are three—SJMP, LJMP and AJMP—which differ in the format of the destination address. JMP is a generic mnemonic which can be used if the programmer does not care which way the jump is encoded.

The SJMP instruction encodes the destination address as a relative offset, as described above. The instruction is 2 bytes long, consisting of the opcode and the relative offset byte. The jump distance is limited to a range of −128 to +127 bytes relative to the instruction following the SJMP.

The LJMP instruction encodes the destination address as a 16-bit constant. The instruction is 3 bytes long, consisting of the opcode and two address bytes. The destination address can be anywhere in the 64K Program Memory space.

The AJMP instruction encodes the destination address as an 11-bit constant. The instruction is 2 bytes long, consisting of the opcode, which itself contains 3 of the 11 address bits, followed by another byte containing the low 8 bits of the destination address. When the instruction is executed, these 11 bits are simply substituted for the low 11 bits in the PC. The high 5 bits stay the same. Hence the destination has to be within the same 2K block as the instruction following the AJMP.

In all cases the programmer specifies the destination address to the assembler in the same way: as a label or as a 16-bit constant. The assembler will put the destination address into the correct format for the given instruction. If the format required by the instruction will not support the distance to the specified destination address, a "Destination out of range" message is written into the List file.

The JMP @A+DPTR instruction supports case jumps. The destination address is computed at execution time as the sum of the 16-bit DPTR register and the Accumulator. Typically, DPTR is set up with the address of a jump table, and the Accumulator is given an index to the table. In a 5-way branch, for example, an integer 0 through 4 is loaded into the Accumulator. The code to be executed might be as follows:

```
MOV     DPTR,#JUMP_TABLE
MOV     A,INDEX_NUMBER
RL      A
JMP     @A+DPTR
```

The RL A instruction converts the index number (0 through 4) to an even number on the range 0 through 8, because each entry in the jump table is 2 bytes long:

```
JUMP_TABLE:
    AJMP     CASE_0
    AJMP     CASE_1
    AJMP     CASE_2
    AJMP     CASE_3
    AJMP     CASE_4
```

Table 4-7 shows a single "CALL addr" instruction, but there are two of them—LCALL and ACALL—which differ in the format in which the subroutine address is given to the CPU. CALL is a generic mnemonic which can be used if the programmer does not care which way the address is encoded.

The LCALL instruction uses the 16-bit address format, and the subroutine can be anywhere in the 64K Program Memory space. The ACALL instruction uses the 11-bit format, and the subroutine must be in the same 2K block as the instruction following the ACALL.

In any case the programmer specifies the subroutine address to the assembler in the same way: as a label or as a 16-bit constant. The assembler will put the address into the correct format for the given instructions.

Subroutines should end with a RET instruction, which returns execution to the instruction following the CALL.

RETI is used to return from an interrupt service routine. The only difference between RET and RETI is that RETI tells the interrupt control system that the interrupt in progress is done. If there is no interrupt in progress at the time RETI is executed, then the RETI is functionally identical to RET.

Table 4-8. Conditional Jumps in 8051 Family Devices

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A $\neq$ 0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A $\neq$ <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> $\neq$ #data		X	X		2

Table 4-8 shows the list of conditional jumps available to the 8051 Family user. All of these jumps specify the destination address by the relative offset method, and so are limited to a jump distance of -128 to $+127$ bytes from the instruction following the conditional jump instruction. Important to note, however, the user specifies to the assembler the actual destination address the same way as the other jumps: as a label or a 16-bit constant.

There is no Zero bit in the PSW. The JZ and JNZ instructions test the Accumulator data for that condition.

The DJNZ instruction (Decrement and Jump if Not Zero) is for loop control. To execute a loop N times, load a counter byte with N and terminate the loop with a DJNZ to the beginning of the loop, as shown below for N = 10:

```
            MOV     COUNTER,#10
    LOOP:   (begin loop)
               *
               *
               *
            (end loop)
            DJNZ    COUNTER,LOOP
            (continue)
```

The CJNE instruction (Compare and Jump if Not Equal) can also be used for loop control as in Figure 4-3. Two bytes are specified in the operand field of the instruction. The jump is executed only if the two bytes are not equal. In the example of Figure 4-3, the two bytes were the data in R1 and the constant 2AH. The initial data in R1 was 2EH. Every time the loop was executed, R1 was decremented, and the looping was to continue until the R1 data reached 2AH.

Another application of this instruction is in "greater than, less than" comparisons. The two bytes in the operand field are taken as unsigned integers. If the first is less than the second, then the Carry bit is set (1). If the first is greater than or equal to the second, then the Carry bit is cleared.

Table 4-9. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Chapter 2, page 2-24

Instructions that Affect Flag Settings[1]

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C,bit	X		
DA	X			ORL C,bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

[1]Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes:

Rn — Register R7–R0 of the currently selected Register Bank.

direct — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].

@Ri — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.

#data — 8-bit constant included in instruction.

#data 16 — 16-bit constant included in instruction.

addr 16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.

bit — Direct Addressed bit in Internal Data RAM or Special Function Register.

* — New operation not provided by 8048AH/8049AH.

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12

Table 4-9. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12

Table 4-9. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24

Mnemonic		Description	Byte	Oscillator Period
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move diret bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24

Table 4-9. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

INSTRUCTION DEFINITIONS

ACALL addr11

Function:	Absolute Call
Description:	ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.
Example:	Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345 H. After executing the instruction,

ACALL   SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes:	2
Cycles:	2

Encoding:

a10 a9 a8 1	0 0 0 1		a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL
$(PC) \leftarrow (PC) + 2$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7\text{-}0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15\text{-}8})$
$(PC_{10\text{-}0}) \leftarrow$ page address

ADD   A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD   A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD   A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	1 r r r

Operation: ADD
(A) ← (A) + (Rn)

ADD   A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 1		direct address

Operation: ADD
(A) ← (A) + (direct)

ADD   A,@Ri

Bytes:	1
Cycles:	1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
$(A) \leftarrow (A) + ((R_i))$

ADD   A,#data

Bytes:	2
Cycles:	1

Encoding:

0 0 1 0	0 1 0 0		immediate data

Operation: ADD
$(A) \leftarrow (A) + \#data$

ADDC   A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC   A,R0

will leave 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

ADDC   A,Rn

     Bytes: 1

    Cycles: 1

  Encoding: | 0 0 1 1 | 1 r r r |

  Operation: ADDC
                $(A) \leftarrow (A) + (C) + (R_n)$

ADDC   A,direct

     Bytes: 2

    Cycles: 1

  Encoding: | 0 0 1 1 | 0 1 0 1 | | direct address |

  Operation: ADDC
                $(A) \leftarrow (A) + (C) + (direct)$

ADDC   A,@Ri

     Bytes: 1

    Cycles: 1

  Encoding: | 0 0 1 1 | 0 1 1 i |

  Operation: ADDC
                $(A) \leftarrow (A) + (C) + ((R_i))$

ADDC   A,#data

     Bytes: 2

    Cycles: 1

  Encoding: | 0 0 1 1 | 0 1 0 0 | | immediate data |

  Operation: ADDC
                $(A) \leftarrow (A) + (C) + \#data$

AJMP   addr11

Function:	Absolute Jump
Description:	AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.
Example:	The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP   JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes:	2
Cycles:	2

Encoding:

a10 a9 a8 0	0 0 0 1		a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP
$(PC) \leftarrow (PC) + 2$
$(PC_{10-0}) \leftarrow$ page address

ANL   <dest-byte>,<src-byte>

Function:	Logical-AND for byte variables
Description:	ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example:	If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL   A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL   P1,#01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL   A,Rn

Bytes: 1

Cycles: 1

Encoding: | 0 1 0 1 | 1 r r r |

Operation: ANL
(A) ← (A) ∧ (Rn)

ANL   A,direct

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 1 | 0 1 0 1 | | direct address |

Operation: ANL

(A) ← (A) ∧ (direct)

ANL   A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 0 1 0 1 | 0 1 1 i |

Operation: ANL

(A) ← (A) ∧ ((Ri))

ANL   A,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 1 | 0 1 0 0 | | immediate data |

Operation: ANL
(A) ← (A) ∧ #data

ANL   direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0		direct address

Operation: ANL
(direct) ← (direct) ∧ (A)

ANL   direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1		direct address		immediate data

Operation: ANL
(direct) ← (direct) ∧ #data

ANL   C,<src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

MOV   C,P1.0   ;LOAD CARRY WITH INPUT PIN STATE

ANL   C,ACC.7   ;AND CARRY WITH ACCUM. BIT 7

ANL   C,/OV   ;AND WITH INVERSE OF OVERFLOW FLAG

ANL   C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0		bit address

Operation: ANL
(C) ← (C) ∧ (bit)

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding: | 1 0 1 1 | 0 0 0 0 | | bit address |

Operation: ANL
(C) ← (C) ∧ ⌐ (bit)

CJNE <dest-byte>,<src-byte>, rel

Function: Compare and Jump if Not Equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
                  CJNE   R7,#60H, NOT__EQ
;                 ...    .....          ;  R7 = 60H.
NOT__EQ:          JC     REQ__LOW        ;  IF R7 < 60H.
;                 ...    .....          ;  R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT__EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE   A,direct,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 0 1 0 1 | | direct address | | rel. address |

Operation: (PC) ← (PC) + 3
IF (A) <> *(direct)*
THEN
       (PC) ← (PC) + *relative offset*

IF (A) < *(direct)*
THEN
       (C) ← 1
ELSE
       (C) ← 0

CJNE   A,#data,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 0 1 0 0 | | immediate data | | rel. address |

Operation: (PC) ← (PC) + 3
IF (A) <> *data*
THEN
       (PC) ← (PC) + *relative offset*

IF (A) < *data*
THEN
       (C) ← 1
ELSE
       (C) ← 0

CJNE   Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 1 r r r | | immediate data | | rel. address |

Operation: (PC) ← (PC) + 3
IF (Rn) <> *data*
THEN
       (PC) ← (PC) + *relative offset*

IF (Rn) < *data*
THEN
       (C) ← 1
ELSE
       (C) ← 0

CJNE   @Ri,#data,rel

Bytes:	3
Cycles:	2

Encoding:

| 1 0 1 1 | 0 1 1 i | | immediate data | | rel. address |

Operation: $(PC) \leftarrow (PC) + 3$
IF $((Ri)) <> data$
THEN
$(PC) \leftarrow (PC) + relative \ offset$

IF $((Ri)) < data$
THEN
$(C) \leftarrow 1$
ELSE
$(C) \leftarrow 0$

CLR   A

Function:	Clear Accumulator
Description:	The Accumulator is cleared (all bits set on zero). No flags are affected.
Example:	The Accumulator contains 5CH (01011100B). The instruction,

CLR   A

will leave the Accumulator set to 00H (00000000B).

Bytes:	1
Cycles:	1

Encoding:

| 1 1 1 1 | 0 1 0 0 |

Operation: CLR
$(A) \leftarrow 0$

CLR   bit

Function:	Clear bit
Description:	The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.
Example:	Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR   P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes:	1
Cycles:	1

Encoding:

1 1 0 0	0 0 1 1

Operation: CLR
(C) ← 0

CLR bit

Bytes:	2
Cycles:	1

Encoding:

1 1 0 0	0 0 1 0		bit address

Operation: CLR
(bit) ← 0

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CPL   A

will leave the Accumulator set to 0A3H (10100011B).

Bytes:	1
Cycles:	1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL
(A) ← ⌐ (A)

CPL   bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly address-able bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

CPL   P1.1

CPL   P1.2

will leave the port set to 5BH (01011011B).

CPL   C

Bytes: 1

Cycles: 1

Encoding: | 1 0 1 1 | 0 0 1 1 |

Operation: CPL
$(C) \leftarrow \neg (C)$

CPL   bit

Bytes: 2

Cycles: 1

Encoding: | 1 0 1 1 | 0 0 1 0 | | bit address |

Operation: CPL
$(bit) \leftarrow \neg (bit)$

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC  A,R3
DA    A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD   A,#99H

DA    A
```

will leave the carry set and 29H in the Accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 − 1 = 29.

Bytes: 1

Cycles: 1

Encoding:	1 1 0 1	0 1 0 0

Operation: DA
-contents of Accumulator are BCD
IF $[[(A_{3-0}) > 9] \lor [(AC) = 1]]$
THEN$(A_{3-0}) \leftarrow (A_{3-0}) + 6$
AND

IF $[[(A_{7-4}) > 9] \lor [(C) = 1]]$
THEN $(A_{7-4}) \leftarrow (A_{7-4}) + 6$

DEC   byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

DEC   @R0

DEC   R0

DEC   @R0

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC   A

Bytes: 1

Cycles: 1

Encoding:	0 0 0 1	0 1 0 0

Operation: DEC
$(A) \leftarrow (A) - 1$

DEC   Rn

Bytes: 1

Cycles: 1

Encoding:	0 0 0 1	1 r r r

Operation: DEC
$(Rn) \leftarrow (Rn) - 1$

DEC direct

Bytes: 2

Cycles: 1

Encoding:

| 0 0 0 1 | 0 1 0 1 | direct address |

Operation: DEC

$(direct) \leftarrow (direct) - 1$

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

| 0 0 0 1 | 0 1 1 i |

Operation: DEC

$((Ri)) \leftarrow ((Ri)) - 1$

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 X 18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

| 1 0 0 0 | 0 1 0 0 |

Operation: DIV

$(A)_{15-8}$
$(B)_{7-0} \leftarrow (A)/(B)$

DJNZ <byte>,<rel-addr>

Function:	Decrement and Jump if Not Zero
Description:	DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ   40H,LABEL__1
DJNZ   50H,LABEL__2
DJNZ   60H,LABEL__3
```

will cause a jump to the instruction at label LABEL__2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
              MOV        R2,#8
TOGGLE:       CPL        P1.7
              DJNZ       R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes:	2
Cycles:	2
Encoding:	`1 1 0 1` `1 r r r` `rel. address`
Operation:	DJNZ

$$(PC) \leftarrow (PC) + 2$$
$$(Rn) \leftarrow (Rn) - 1$$
IF   (Rn) > 0 or (Rn) < 0
    THEN
           $(PC) \leftarrow (PC) + rel$

DJNZ   direct,rel

Bytes:	3
Cycles:	2

Encoding:

1 1 0 1	0 1 0 1		direct address		rel. address

Operation:

DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) − 1
IF (direct) > 0 or (direct) < 0
    THEN
            (PC) ← (PC) + rel

INC   <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

INC   @R0
INC   R0
INC   @R0

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC   A

Bytes:	1
Cycles:	1

Encoding:

0 0 0 0	0 1 0 0

Operation:

INC
(A) ← (A) + 1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC

$(Rn) \leftarrow (Rn) + 1$

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC

$(direct) \leftarrow (direct) + 1$

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC

$((Ri)) \leftarrow ((Ri)) + 1$

INC DPTR

Function:	Increment Data Pointer
Description:	Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example:	Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC   DPTR
INC   DPTR
INC   DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes:	1
Cycles:	2

Encoding:		1 0 1 0	0 0 1 1	

Operation:	INC (DPTR) ← (DPTR) + 1

JB bit,rel

Function:	Jump if Bit set
Description:	If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.
Example:	The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

```
JB   P1.2,LABEL1

JB   ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes:	3
Cycles:	2

Encoding:		0 0 1 0	0 0 0 0		bit address		rel. address	

Operation:	JB
	(PC) ← (PC) + 3
	IF (bit) = 1
	    THEN
	        (PC) ← (PC) + rel

JBC   bit,rel

Function:	Jump if Bit is set and Clear bit
Description:	If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC   ACC.3,LABEL1
JBC   ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

| 0 0 0 1 | 0 0 0 0 | | bit address | | rel. address |

Operation: JBC
$(PC) \leftarrow (PC) + 3$
IF   (bit) = 1
    THEN
        (bit) $\leftarrow$ 0
        $(PC) \leftarrow (PC) + rel$

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL 2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0		rel. address

Operation: JC
$$(PC) \leftarrow (PC) + 2$$
IF (C) = 1
       THEN
              $$(PC) \leftarrow (PC) + rel$$

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP__TBL:

```
              MOV    DPTR,#JMP__TBL
              JMP    @A+DPTR
JMP__TBL:     AJMP   LABEL0
              AJMP   LABEL1
              AJMP   LABEL2
              AJMP   LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB   P1.3,LABEL1
JNB   ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding: | 0 0 1 1 | 0 0 0 0 | | bit address | | rel. address |

Operation: JNB
$(PC) \leftarrow (PC) + 3$
IF   (bit) = 0
      THEN $(PC) \leftarrow (PC) + rel.$

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding: | 0 1 0 1 | 0 0 0 0 | | rel. address |

Operation: JNC
$(PC) \leftarrow (PC) + 2$
IF   (C) = 0
      THEN $(PC) \leftarrow (PC) + rel$

JNZ rel

Function:	Jump if Accumulator Not Zero
Description:	If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Example:	The Accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   A
JNZ   LABEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0		rel. address

Operation: JNZ
(PC) ← (PC) + 2
IF   (A) ≠ 0
    THEN   (PC) ← (PC) + rel

JZ rel

Function:	Jump if Accumulator Zero
Description:	If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Example:	The Accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC A
JZ    LABEL2
```

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0		rel. address

Operation: JZ
(PC) ← (PC) + 2
IF   (A) = 0
    THEN   (PC) ← (PC) + rel

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

LCALL   SUBRTN

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

Bytes: 3

Cycles: 2

Encoding:

| 0 0 0 1 | 0 0 1 0 | addr15-addr8 | addr7-addr0 |

Operation: LCALL
$(PC) \leftarrow (PC) + 3$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7-0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15-8})$
$(PC) \leftarrow addr_{15-0}$

LJMP addr16

Function:	Long Jump
Description:	LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.
Example:	The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP   JMPADR

at location 0123H will load the program counter with 1234H.

Bytes:	3
Cycles:	2

Encoding:

0 0 0 0	0 0 1 0		addr15-addr8		addr7-addr0

Operation: LJMP
$(PC) \leftarrow addr_{15-0}$

MOV   <dest-byte>,<src-byte>

Function:	Move byte variable
Description:	The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.
	This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.
Example:	Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV   R0,#30H   ;R0 < = 30H
MOV   A,@R0     ;A < = 40H
MOV   R1,A      ;R1 < = 40H
MOV   R,@R1     ;B < = 10H
MOV   @R1,P1    ;RAM (40H) < = 0CAH
MOV   P2,P1     ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV   A,Rn

Bytes:	1
Cycles:	1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

MOV   A,direct

Bytes:	2
Cycles:	1

Encoding:

1 1 1 0	0 1 0 1		direct address

Operation: MOV
(A) ← (direct)

MOV A,ACC is not a valid instruction.

MOV   A,@Ri

Bytes:	1
Cycles:	1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV   A,#data

Bytes:	2
Cycles:	1

Encoding:

0 1 1 1	0 1 0 0		immediate data

Operation: MOV
(A) ← #data

MOV   Rn,A

Bytes: 1

Cycles: 1

Encoding: | 1 1 1 1 | 1 r r r |

Operation: MOV
(Rn) ← (A)

MOV   Rn,direct

Bytes: 2

Cycles: 2

Encoding: | 1 0 1 0 | 1 r r r | | direct addr. |

Operation: MOV
(Rn) ← (direct)

MOV   Rn,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 1 | 1 r r r | | immediate data |

Operation: MOV
(Rn) ← #data

MOV   direct,A

Bytes: 2

Cycles: 1

Encoding: | 1 1 1 1 | 0 1 0 1 | | direct address |

Operation: MOV
(direct) ← (A)

MOV   direct,Rn

Bytes: 2

Cycles: 2

Encoding: | 1 0 0 0 | 1 r r r | | direct address |

Operation: MOV
(direct) ← (Rn)

MOV   direct,direct

  Bytes: 3

  Cycles: 2

  Encoding: | 1 0 0 0 | 0 1 0 1 | | dir. addr. (src) | | dir. addr. (dest) |

  Operation: MOV
  (direct) ← (direct)

MOV   direct,@Ri

  Bytes: 2

  Cycles: 2

  Encoding: | 1 0 0 0 | 0 1 1 i | | direct addr. |

  Operation: MOV
  (direct) ← ((Ri))

MOV   direct,#data

  Bytes: 3

  Cycles: 2

  Encoding: | 0 1 1 1 | 0 1 0 1 | | direct address | | immediate data |

  Operation: MOV
  (direct) ← #data

MOV   @Ri,A

  Bytes: 1

  Cycles: 1

  Encoding: | 1 1 1 1 | 0 1 1 i |

  Operation: MOV
  ((Ri)) ← (A)

MOV   @Ri,direct

  Bytes: 2

  Cycles: 2

  Encoding: | 1 0 1 0 | 0 1 1 i | | direct addr. |

  Operation: MOV
  ((Ri)) ← (direct)

MOV   @RI, # data

Bytes:	2
Cycles:	1

Encoding:

0 1 1 1	0 1 1 i		immediate data

Operation: MOV
((RI)) ← #data

MOV   <dest-bit>, <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

MOV   P1.3,C
MOV   C,P3.3
MOV   P1.2,C

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV   C,bit

Bytes:	2
Cycles:	1

Encoding:

1 0 1 0	0 0 1 0		bit address

Operation: MOV
(C) ← (bit)

MOV   bit,C

Bytes:	2
Cycles:	2

Encoding:

1 0 0 1	0 0 1 0		bit address

Operation: MOV
(bit) ← (C)

MOV   DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV   DPTR,#1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

| 1 0 0 1 | 0 0 0 0 | | immed. data15-8 | | immed. data7-0 |
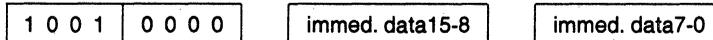
Operation: MOV
$(DPTR) \leftarrow \#data_{15-0}$
$DPH \, \square \, DPL \leftarrow \#data_{15-8} \, \square \, \#data_{7-0}$

MOVC A,@A + <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL__PC:  INC    A

          MOVC   A,@A + PC

          RET

          DB     66H

          DB     77H

          DB     88H

          DB     99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A + DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
$(A) \leftarrow ((A) + (DPTR))$

MOVC   A,@A + PC

Bytes: 1

Cycles: 2

Encoding:

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

MOVX   <dest-byte>,<src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

MOVX   A,@R1

MOVX   @R0,A

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX   A,@Ri

     Bytes: 1

     Cycles: 2

   Encoding: | 1 1 1 0 | 0 0 1 i |

   Operation: MOVX
                      (A) ← ((Ri))

MOVX   A,@DPTR

     Bytes: 1

     Cycles: 2

   Encoding: | 1 1 1 0 | 0 0 0 0 |

   Operation: MOVX
                      (A) ← ((DPTR))

MOVX   @Ri,A

     Bytes: 1

     Cycles: 2

   Encoding: | 1 1 1 1 | 0 0 1 i |

   Operation: MOVX
                      ((Ri)) ← (A)

MOVX   @DPTR,A

     Bytes: 1

     Cycles: 2

   Encoding: | 1 1 1 1 | 0 0 0 0 |

   Operation: MOVX
                      (DPTR) ← (A)

NOP

Function:	No Operation
Description:	Execution continues at the following instruction. Other than the PC, no registers or flags are affected.
Example:	It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

CLR   P2.7

NOP

NOP

NOP

NOP

SETB   P2.7

Bytes:	1
Cycles:	1

Encoding:

0 0 0 0	0 0 0 0

Operation: NOP

$(PC) \leftarrow (PC) + 1$

MUL   AB

Function:	Multiply
Description:	MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otehrwise it is cleared. The carry flag is always cleared.
Example:	Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL   AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes:	1
Cycles:	4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL

$(A)_{7-0} \leftarrow (A) \times (B)$

$(B)_{15-8}$

ORL   \<dest-byte\> \<src-byte\>

Function:	Logical-OR for byte variables
Description:	ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL   A,R0

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

ORL   P1,#00110010B

will set bits 5, 4, and 1 of output Port 1.

ORL   A,Rn

Bytes:	1
Cycles:	1
Encoding:	`0 1 0 0` `1 r r r`
Operation:	ORL (A) ← (A) ∨ (Rn)

ORL   A,direct

  Bytes: 2

  Cycles: 1

  Encoding: | 0 1 0 0 | 0 1 0 1 | | direct address |

  Operation: ORL
  (A) ← (A) ∨ (direct)

ORL   A,@Ri

  Bytes: 1

  Cycles: 1

  Encoding: | 0 1 0 0 | 0 1 1 i |

  Operation: ORL
  (A) ← (A) ∨ ((Ri))

ORL   A,#data

  Bytes: 2

  Cycles: 1

  Encoding: | 0 1 0 0 | 0 1 0 0 | | immediate data |

  Operation: ORL
  (A) ← (A) ∨ #data

ORL   direct,A

  Bytes: 2

  Cycles: 1

  Encoding: | 0 1 0 0 | 0 0 1 0 | | direct address |

  Operation: ORL
  (direct) ← (direct) ∨ (A)

ORL   direct,#data

  Bytes: 3

  Cycles: 2

  Encoding: | 0 1 0 0 | 0 0 1 1 | | direct addr. | | immediate data |

  Operation: ORL
  (direct) ← (direct) ∨ #data

ORL   C, <src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise . A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

MOV   C,P1.0   ;LOAD CARRY WITH INPUT PIN P10

ORL   C,ACC.7   ;OR CARRY WITH THE ACC. BIT 7

ORL   C,/OV   ;OR CARRY WITH THE INVERSE OF OV.

ORL   C,bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0		bit address

Operation: ORL
$(C) \leftarrow (C) \vee (bit)$

ORL   C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0		bit address

Operation: ORL
$(C) \leftarrow (C) \vee (\overline{bit})$

POP   direct

Function: Pop from stack.

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

POP   DPH

POP   DPL

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

POP   SP

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0		direct address

Operation: POP
(direct) ← ((SP))
(SP) ← (SP) − 1

PUSH direct

Function:	Push onto stack
Description:	The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.
Example:	On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

PUSH   DPL

PUSH   DPH

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes:	2
Cycles:	2

Encoding:

1 1 0 0	0 0 0 0		direct address

Operation: PUSH
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (direct)$

RET

Function:	Return from subroutine
Description:	RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.
Example:	The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes:	1
Cycles:	2

Encoding:

0 0 1 0	0 0 1 0

Operation: RET
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RETI

Function:	Return from interrupt
Description:	RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.
Example:	The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes:	1
Cycles:	2

Encoding:

0 0 1 1	0 0 1 0

Operation: RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RL A

Function:	Rotate Accumulator Left
Description:	The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes:	1
Cycles:	1

Encoding:

0 0 1 0	0 0 1 1

Operation: RL
$(A_n + 1) \leftarrow (An)$ $n = 0 - 6$
$(A0) \leftarrow (A7)$

RLC   A

Function:	Rotate Accumulator Left through the Carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC   A

leaves the Accumulator holding the value 8BH (10001010B) with the carry set.

Bytes:	1
Cycles:	1

Encoding:

0 0 1 1	0 0 1 1

Operation: RLC
$(A_n + 1) \leftarrow (A_n)$ $n = 0 - 6$
$(A_0) \leftarrow (C)$
$(C) \leftarrow (A_7)$

RR   A

Function:	Rotate Accumulator Right
Description:	The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B). The instruction,

RR   A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes:	1
Cycles:	1

Encoding:

0 0 0 0	0 0 1 1

Operation: RR
$(A_n) \leftarrow (A_n + 1)$ $n = 0 - 6$
$(A_7) \leftarrow (A_0)$

RRC   A

Function:	Rotate Accumulator Right through Carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

RRC   A

leaves the Accumulator holding the value 62 (01100010B) with the carry set.

Bytes:	1
Cycles:	1

Encoding:	0 0 0 1	0 0 1 1

Operation: RRC
$(An) \leftarrow (An + 1)$ $n = 0 - 6$
$(A7) \leftarrow (C)$
$(C) \leftarrow (A0)$

SETB   <bit>

Function:	Set Bit
Description:	SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.
Example:	The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

SETB   C

SETB   P1.0

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB   C

Bytes:	1
Cycles:	1

Encoding:	1 1 0 1	0 0 1 1

Operation: SETB
$(C) \leftarrow 1$

SETB   bit

Bytes:	2
Cycles:	1

Encoding:

1 1 0 1	0 0 1 0		bit address

Operation: SETB
(bit) ← 1

SJMP   rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

SJMP   RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes:	2
Cycles:	2

Encoding:

1 0 0 0	0 0 0 0		rel. address

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

SUBB A,<src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should not be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

Operation: SUBB
(A) ← (A) − (C) − (Rn)

SUBB A,direct

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A) ← (A) − (C) − (direct)

SUBB   A,@RI

Bytes:	1

Cycles:	1

Encoding: | 1 0 0 1 | 0 1 1 i |

Operation: SUBB
$(A) \leftarrow (A) - (C) - ((Ri))$

SUBB   A,#data

Bytes:	2

Cycles:	1

Encoding: | 1 0 0 1 | 0 1 0 0 | | immediate data |

Operation: SUBB
$(A) \leftarrow (A) - (C) - \#data$

SWAP   A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP   A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes:	1

Cycles:	1

Encoding: | 1 1 0 0 | 0 1 0 0 |

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A,<byte>

Function:	Exchange Accumulator with byte variable
Description:	XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.
Example:	R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH   A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH   A,Rn

Bytes:	1
Cycles:	1

Encoding:

1 1 0 0	1 r r r

Operation: XCH
$(A) \rightleftarrows (Rn)$

XCH   A,direct

Bytes:	2
Cycles:	1

Encoding:

1 1 0 0	0 1 0 1		direct address

Operation: XCH
$(A) \rightleftarrows (direct)$

XCH   A,@Ri

Bytes:	1
Cycles:	1

Encoding:

1 1 0 0	0 1 1 i

Operation: XCH
$(A) \rightleftarrows ((Ri))$

XCHD A,@Ri

Function:	Exchange Digit
Description:	XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.
Example:	R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD   A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the Accumulator.

Bytes:	1
Cycles:	1
Encoding:	$\boxed{1\ 1\ 0\ 1}\ \boxed{0\ 1\ 1\ i}$
Operation:	XCHD $(A_{3-0}) \rightleftarrows ((Ri_{3-0}))$

XRL   <dest-byte>,<src-byte>

Function:	Logical Exclusive-OR for byte variables
Description:	XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

Example:	If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL   A,R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL   P1,#00110001B

will complement bits 5, 4, and 0 of output Port 1.

XRL   A,Rn

Bytes: 1

Cycles: 1

Encoding: | 0 1 1 0 | 1 r r r |

Operation: XRL
(A) ← (A) ∀ (Rn)

XRL   A,direct

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 0 | 0 1 0 1 | | direct address |

Operation: XRL
(A) ← (A) ∀ (direct)

XRL   A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 0 1 1 0 | 0 1 1 i |

Operation: XRL
(A) ← (A) ∀ ((Ri))

XRL   A,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 0 | 0 1 0 0 | | immediate data |

Operation: XRL
(A) ← (A) ∀ #data

XRL   direct,A

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 0 | 0 0 1 0 | | direct address |

Operation: XRL
(direct) ← (direct) ∀ (A)

XRL   direct,#data

Bytes:	3
Cycles:	2

Encoding:

| 0 1 1 0 | 0 0 1 1 | | direct address | | immediate data |

Operation: XRL
(direct) ← (direct) ∀ #data

CHAPTER 5

Software Routines

Chapter 5 contains two sections:

- 8051 Programming Techniques
- Peripheral Interfacing Techniques.

The first section has 8051 software examples for some common routines in controller applications. Some routines included are multiple-precision arithmetic and table look-up techniques.

Peripheral Interfacing Techniques include routines for handling the 8051's I/O ports, serial channel and timer/counters. Discussed in this section is I/O port reconfiguration, software delay timing, and transmitting serial port character strings along with other routines.

8051 PROGRAMMING TECHNIQUES

Radix Conversion Routines

The divide instruction can be used to convert a number from one radix to another. BINBCD is a short subroutine to convert an 8-bit unsigned binary integer in the accumulator (between 0 & 255) to a 3-digit (2 byte) BCD representation. The hundred's digit is returned in one variable (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

```
;
;BINBCD      CONVERT 8-BIT BINARY VARIABLE IN ACCUMULATOR
;            TO 3-DIGIT PACKED BCD FORMAT.
;            HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
;            TENS' AND ONES' PLACES IN 'TENONE'.
;
HUND         DATA   21H
TENONE       DATA   22H
;
BINBCD:      MOV    B,#100        ;DIVIDED BY 100 TO
             DIV    AB            ;DETERMINE NUMBER OF HUNDREDS
             MOV    HUND,A
             MOV    A,#10         ;DIVIDE REMAINDER BY TEN TO
             XCH    A,B           ;DETERMINE NUMBER OF TENS LEFT
             DIV    AB            ;TEN'S DIGIT IN ACC, REMAINDER IS
                                  ;ONE'S DIGIT
             SWAP   A
             ADD    A,B           ;PACK BCD DIGITS IN ACC
             MOV    TENONE,A
             RET
;
```

The divide instruction can also separate data in the accumulator into sub-fields. For example, dividing packed BCD data by 16 will separate the two nibbles, leaving the high-order digit in the accumulator and the low-order digit (remainder) in B. Each is right-justified, so the digits can be processed individually. This example receives two packed BCD digits in the accumulator, separates the digits, computes their product, and returns the product in packed BCD format in the accumulator.

```
;
;MULBCD      UNPACK TWO BCD DIGITS RECEIVED IN ACCUMULATOR
;            FIND THEIR PRODUCT, AND RETURN PRODUCT
;            IN PACKED BCD FORMAT IN ACCUMULATOR
;
;MULBCD:     MOV    B,#10H        ;DIVIDE INPUT BY 16
             DIV    AB            ;A & B HOLD SEPARATED DIGITS
                                  ;(EACH RIGHT JUSTIFIED IN REGISTER).
             MUL    AB            ;A HOLDS PRODUCT IN BINARY FORMAT
                                  ;(0 TO 99 (DECIMAL) = 0 TO 63H)
             MOV    B,#10         ;DIVIDE PRODUCT BY 10
             DIV    AB            ;A HOLDS NUMBER OF TENS, B HOLDS
                                  ;REMAINDER
```

```
            SWAP    A
            ORL     A,B                     ;PACK DIGITS
            RET
```

Multiple Precision Arithmetic

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple-precision calculations by repeating the operation with successively higher-order operand bytes. If the input data for a multiple-precision operation is an unsigned string of integers, the carry flag will be set upon completion if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data, the most significant bit of the original input data's most significant byte indicates the sign of the string, so the overflow flag (OV) will indicate if overflow or underflow occurred.

```
;
;SUBSTR     SUBTRACT STRING INDICATED BY R1
;           FROM STRING INDICATED BY R0 TO
;           PRECISION INDICATED BY R2.
;           CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR:     CLR     C                       ;BORROW = 0.
SUBS1:      MOV     A,@R0                   ;LOAD MINUEND BYTE
            SUBB    A,@R1                   ;SUBTRACT SUBTRAHEND BYTE
            MOV     @R0,A                   ;STORE DIFFERENCE BYTE
            INC     R0                      ;BUMP POINTERS TO NEXT PLACE
            INC     R1
            DJNZ    R2,SUBS1                ;LOOP UNTIL DONE
;
;           WHEN DONE, TEST IF OVERFLOW OCCURRED
;           ON LAST ITERATION OF LOOP.
;
            JNB     OV,OV_OK
;           ...     ......                  (OVERFLOW RECOVERY ROUTINE)
OV·OK:      RET                             ;RETURN
```

Table Look-Up Sequences

The two versions of the MOVC instructions are used as part of a 3-step sequence to access look-up tables in ROM. To use the DPTR version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVC A, @A + DPTR. The data pointer may be loaded with a constant for short tables, or to allow more complicated data structures, and tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-based version is used with smaller, "local" tables, and has the advantage of not affecting the data pointer. This makes it useful in interrupt routines or other situations where the DPTR contents might be significant. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction's address to the start of the table by adding that offset to the accumulator; then execute the MOVC A,@A + PC instruction.

As a non-trivial situation where this instruction would be used, consider applications which store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in the linear (one-dimensional) program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

Entry Address = [BASE + (NDIMEN x INDEXI) + INDEXJ]

The subroutine MATRX1 can access an entry in any array with less than 255 elements, e.g., an 11x21 array with 231 elements. The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

To handle the more general case, subroutine MATRX2 allows tables to be unlimited in size, by combining the MUL instruction, double-precision addition, and the data pointer-based version of MOVC. The only restriction is that each index be between 0 and 255.

```
;
;MATRX      LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;           TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;           USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC A,@A + PC'.
;           THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;           BE SMALL, I.E. LESS THAN ABOUT 255 ENTRIES.
;           TABLE USED IN THIS EXAMPLE IS 11 x 21.
;           DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;
;           [(BASE ADDRESS) + (21 X INDEXI) + (INDEXJ)]
;
INDEXI      EQU    R6                     ;FIRST COORDINATE OF ENTRY (0-10).
INDEXJ      DATA   23H                    ;SECOND COORDINATE OF ENTRY (0-20).
;
MATRX1:     MOV    A,INDEXI
            MOV    B,#21
            MUL    AB                     ;(21 X INDEXI)
            ADD    A,INDEXJ               ;ADD IN OFFSET WITHIN ROW
;
;           ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;           ENTRY (0,0).
;
            INC    A
            MOVC   A,@A + PC
            RET
BASE1:      DB     1                      ;(entry 0,0)
            DB     2                      ;(entry 0,1)
;           ...    ......
            DB     21                     ;(entry 0,20)
            DB     22                     ;(entry 1,0)
;           ...    ......
            DB     42                     ;(entry 1,20)
;           ...    ......
;           ...    ......
            DB     231                    ;(entry 10,20)
MATRX2:     MOV    A,INDEXI               ;LOAD FIRST COORDINATE
            MOV    B,#NDIMEN
            MUL    AB                     ;INDEXI X NDIMEN
            ADD    A,#LOW(BASE2)          ;ADD IN 16-BIT BASE ADDRESS
            MOV    DPL,A
            MOV    A,B
            ADDC   A,#HIGH(BASE2)
            MOV    DPH,A                  ;DPTR=(BASE ADDR) + (INDEXI + NDIMEN)
            MOV    A,INDEXJ
            MOVC   A,@A + DPTR            ;ADD INDEXJ AND FETCH BYTE
            RET
;           ...    ......
BASE2       DB     0                      ;(entry 0,0)
            DB     0                      ;(entry 0,1)
;           ...    ......
            DB     0                      ;(entry 0, NDIMEN-1)
            DB     0                      ;(entry 1,0)
;           ...    ......
            DB     0                      ;(entry 1, NDIMEN-1)
;           ...    ......
;           ...    ......
            DB     0                      ;(entry MDIMEN-1, NDIMEN-1)
```

Saving CPU Status During Interrupts

When the 8051 hardware recognizes an interrupt request, program control branches automatically to the corresponding service routine, by forcing the CPU to process a Long CALL (LCALL) instruction to the appropriate address. The re turn address is stored on the top of the stack. After completing the service routine, an RETI instruction returns the processor to the background program at the point from which it was interrupted.

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error. An example of this will be given later in this section, in the second method of I/O port reconfiguration.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save such registers on the stack.

```
        ;
        ;LOC_TMP    EQU     $                   ;REMEMBER LOCATION COUNTER
        ;
                    ORG     0003H               ;STARTING ADDRESS FOR INTERRUPT ROUTINE
                    LJMP    SERVER              ;JUMP TO ACTUAL SERVICE ROUTINE LOCATE
                                                ;ELSEWHERE
        ;           ...     ......
                    ORG     LOC_TMP             ;RESTORE LOCATION COUNTER
        SERVER:     PUSH    PSW
                    PUSH    ACC                 ;SAVE ACCUMULATOR (NOTE DIRECT ADDRESS
                                                ;NOTATION)
                    PUSH    B                   ;SAVE B REGISTER
                    PUSH    DPL                 ;SAVE DATA POINTER
                    PUSH    DPH                 ;
                    MOV     PSW,#00001000B      ;SELECT REGISTER BANK 1
        ;           ...     ......
        ;           ...     ......
                    POP     DPH                 ;RESTORE REGISTERS IN REVERSE ORDER
                    POP     DPL
                    POP     B
                    POP     ACC
                    POP     PSW                 ;RESTORE PSW AND RE-SELECT ORIGINAL
                                                ;REGISTER BANK
                    RETI                        ;RETURN TO MAIN PROGRAM AND RESTORE
                                                ;INTERRUPT LOGIC
```


Figure 5-1. Stack Contents During Interrupt
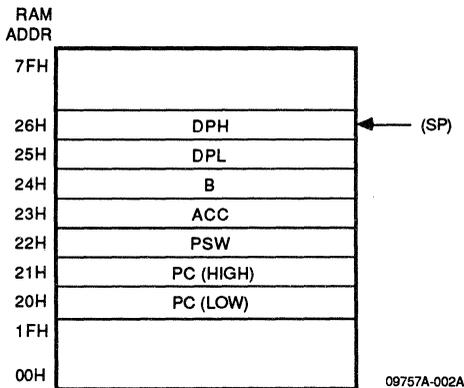
If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 5-1; SP would contain 26H. This is the most general case; if the service routine doesn't alter the B-register and data pointer, for example, the instruction saving and restoring those registers could be omitted.

Passing Parameters on the Stack

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer, or simply pop the stack into registers before processing.

```
HEXASC:    MOV    R0,SP
           DEC    R0                   ;ACCESS LOCATION PARAMETER PUSHED ONTO
           DEC    R0                   ;STACK
           XCH    A,@R0                ;READ INPUT PARAMETER AND SAVE
                                       ;ACCUMULATOR
           ANL    A,#0FH               ;MASK ALL BUT LOW-ORDER 4 BITS
           ADD    A,#2                 ;ALLOW FOR OFFSET FROM MOVC TO TABLE
           MOVC   A,@A + PC            ;READ LOOK-UP TABLE ENTRY
           XCH    A,@R0                ;PASS BACK TRANSLATED VALUE AND RESTORE
                                       ;ACCUMULATOR
           RET                         ;RETURN TO BACKGROUND PROGRAM
ASCTBL:    DB     '0'                  ;ASCII CODE FOR 00H
           DB     '1'                  ;ASCII CODE FOR 01H
           DB     '2'                  ;ASCII CODE FOR 02H
           DB     '3                   ;ASCII CODE FOR 03H
           DB     '4'                  ;ASCII CODE FOR 04H
           DB     '5'                  ;ASCII CODE FOR 05H
           DB     '6'                  ;ASCII CODE FOR 06H
           DB     '7'                  ;ASCII CODE FOR 07H
           DB     '8'                  ;ASCII CODE FOR 08H
           DB     '9'                  ;ASCII CODE FOR 09H
           DB     'A'                  ;ASCII CODE FOR 0AH
           DB     'B'                  ;ASCII CODE FOR 0BH
           DB     'C'                  ;ASCII CODE FOR 0CH
           DB     'D'                  ;ASCII CODE FOR 0DH
           DB     'E'                  ;ASCII CODE FOR 0EH
           DB     'F'                  ;ASCII CODE FOR 0FH
```

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the subroutine HEXASC converts a hexadecimal value to ASCII code for its low-order digit. It first reads a parameter stored on the stack by the calling program, then uses the low-order bits to access a local 16-entry look-up table holding ASCII codes, stores the appropriate code back in the stack and then returns. The accumulator contents are left unchanged.

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result to any destination register or port later. There is even the option of leaving a value on the stack if it won't be needed until later. The example below converts the three-digit BCD value computed in the Radix Conversion example above to a three-character string, calling a subroutine SP_OUT to output an 8-bit code in the accumulator.

```
;             ...    ......
           PUSH   HUND
           CALL   HEXASC               ;CONVERT HUNDREDS DIGIT
           POP    ACC
           CALL   SP_OUT               ;TRANSMIT HUNDREDS CHARACTER
           PUSH   TENONE
           CALL   HEXASC               ;CONVERT ONE'S PLACE DIGIT
                                       ;BUT LEAVE ON STACK!
           MOV    A,TENONE
           SWAP   A                    ;RIGHT-JUSTIFY TEN'S PLACE
           PUSH   ACC                  ;CONVERT TEN'S PLACE DIGIT
           CALL   HEXASC
           POP    ACC
           CALL   SP_OUT               ;TRANSMIT TEN'S PLACE CHARACTER
           POP    ACC
           CALL   SP_OUT               ;TRANSMIT ONE'S PLACE CHARACTER
;             ...    ......
```

N-Way Branching

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, fixed at assembly time.) Each has advantages for different applications.

In a typical N-way branch situation, the potential destinations are generally known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of offset values in ROM to indicate the relative starting addresses of the several routines.

JMP @A + DPTR is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the 8-bit unsigned accumulator contents with the contents of the 16-bit data pointer, just like MOV A,@A + DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a 16-bit addition is performed: a carry-out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types (and sizes) of buffer memory for different speeds and options.

```
;
;MEMSEL      EQU     R3
;
JUMP_4:      MOV     A,MEMSEL
             MOV     DPTR,#JMPTBL
             MOVC    A,@A + DPTR
             JMP     @A + DPTR
JMPTBL:      DB      MEMSP0-JMPTBL
             DB      MEMSP1-JMPTBL
             DB      MEMSP2-JMPTBL
             DB      MEMSP3-JMPTBL
MEMSP0:      MOV     A,@R0           ;READ FROM INTERNAL RAM
             RET
MEMSP1:      MOVX    A,@R0           ;READ FROM 256 BYTE EXTERNAL RAM
             RET
MEMSP2:      MOV     DPL,R0          ;READ 64K BYTE EXTERNAL RAM
             MOV     DPH,R1
             MOVX    A,@DPTR
             RET
MEMSP3:      MOV     A,R1            ;READ 4K BYTE EXTERNAL RAM
             ANL     A,#07H
             ANL     P1,#11111000B
             ORL     P1,A
             MOVX    A,@R0
             RET
```

To use this approach, the size of the jump table plus the length of the alternate routines must be less than 256 bytes. The jump table and routines may be located anywhere in program memory and are independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all residing in the same 2K page of program memory, the following technique may be used. In the printing terminal example, this sequence could process 128 different codes for ASCII characters arriving via the 8051 serial port.

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

```
;
OPTION       EQU     R3
;            ...     ......
;            ...     ......
JMP128:      MOV     A,OPTION
             RL      A               ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
             MOV     DPTR,#INSTBL    ;FIRST ENTRY IN JUMP TABLE
             JMP     @A + DPTR       ;JUMP INTO JUMP TABLE
;            ...     ......
INSTBL:      AJMP    PROC00          ;128 CONSECUTIVE
             AJMP    PROC01          ;AJMP INSTRUCTIONS
             AJMP    PROC02
;            ...     ......
;            ...     ......
             AJMP    PROC7E
             AJMP    PROC7F
```

Computing Branch Destinations at Run Time

In some rare situations, 128 options are insufficient, the destination routines may cross a 2K page boundary, or a branch destination is not known at assembly time (for whatever reason), and therefore cannot be easily included in the assembled code. These situations can all be handled by computing the destination address at runtime with standard arithmetic or table look-up instructions, then performing an indirect branch to that address.

There are two simple ways to execute this last step, assuming the 16-bit destination address has already been computed. The first is to load the address into the DPH and DPL registers, clear the accumulator and branch using the JMP @A + DPTR instruction; the second is to push the destination address onto the stack, low-order byte first (so as to mimic a call instruction) then pop that address into the PC by performing a return instruction. This also adjusts the stack pointer to its previous value. The code segment below illustrates the latter possibility.

```
;
RTEMP        EQU     R7
;            ...     ......
JMP256:      MOV     DPTR,#ADRTBL    ;FIRST ADDRESS TABLE ENTRY
             MOV     A,OPTION        ;LOAD INDEX INTO TABLE
             CLR     C
             RLC     A               ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
             JNC     LOW128
             INC     DPH             ;FIX BASE IF INDEX >127.
LOW128:      MOV     RTEMP,A         ;SAVE ADJUSTED ACC FOR SECOND READ
             INC     A               ;READ LOW-ORDER BYTE FIRST
             MOVC    A,@A + DPTR     ;GET LOW-ORDER BYTE FROM TABLE
             PUSH    ACC
             MOV     A,RTEMP         ;RELOAD ADJUSTED ACC
             MOVC    A,@A + DPTR     ;GET HIGH-ORDERED BYTE FROM TABLE
             PUSH    ACC
;
;            THE TWO ACC PUSHES HAVE PRODUCED
;            A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
;            TO THE DESIRED STARTING ADDRESS.
;            IT MAY BE REACHED BY POPPING THE STACK
;            INTO THE PC.
             RET
;            ...     ......
;            ...     ......
;            ...     ......
ADRTBL:      DW      PROC00          ;UP TO 256 CONSECUTIVE DATA
             DW      PROC01          ;WORDS INDICATING STARTING ADDRESSES
;            ...     ......
;            ...     ......
             DW      PROCFF
;
```

In-Line-Code Parameter-Passing

Parameters can be passed by loading appropriate registers with values before calling the subroutine. This technique is inefficient if a lot of the parameters are constants, since each would require a separate register to carry it, and a separate instruction to load the register each time the routine is called.

If the routine is called frequently, a more code-efficient way to transfer constants is "in-line-code" parameter-passing. The constants are actually part of the program code, immediately following the call instruction. The subroutine determines where to find them from the return address on the stack, and then reads the parameters it needs from program memory.

For example, assume a utility named ADD-BCD adds a 16-bit packed-BCD constant with a 2-byte BCD variable in internal RAM and stores the sum in a different 2-byte buffer. The utility must be given the constant and both buffer addresses. Rather than using four working registers to carry this information, all 4 bytes could be inserted into program memory each time the utility is called. Specifically, the calling sequence below invokes the utility to add 1234 (decimal) with the string at internal RAM address 56H, and store the sum in a buffer at location 78H.

The ADDBCD subroutine determines at what point the call was made by popping the return address from the stack into the data pointer high- and low-order bytes. A MOVC instruction then reads the parameters from program memory as they are needed. When done, ADDBCD resumes execution by jumping to the instruction following the last parameter.

```
;           ...      ......
            CALL     ADDBCD
            DW       1234H              ;BCD CONSTANT
            DB       56H               ;SOURCE STRING ADDRESS
            DB       78H               ;DESTINATION STRING ADDRESS
;           ...      ......            ;CONTINUATION OF PROGRAM
;
;
;
ADDBCD:     POP      DPH               ;POP RETURN ADDRESS INTO DPTR
            POP      DPL
            MOV      A,#2              ;INDEX FOR SOURCE STRING PARAMETER
            MOVC     A,@A + DPTR       ;GET SOURCE STRING LOCATION
            MOV      R0,A
            MOV      A,#3              ;INDEX FOR DESTINATION STRING PARAMETER
            MOVC     A,@A + DPTR       ;GET DESTINATION ADDRESS
            MOV      R1,A
            MOV      A,#1              ;INDEX FOR 16-BIT CONSTANT LOW BYTE
            MOVC     A,@A + DPTR       ;GET LOW-ORDER VALUE
            ADD      A,@R0             ;COMPUTE LOW-ORDER BYTE OF SUM
            DA       A                 ;DECIMAL ADJUST FOR ADDITION
            MOV      @R1,A             ;SAVE IN BUFFER
            INC      R0
            INC      R1
            CLR      A                 ;INDEX FOR HIGH-BYTE = 0
            MOVC     A,@A + DPTR       ;GET HIGH-ORDER CONSTANT
            ADDC     A,@R0
            DA       A                 ;DECIMAL ADJUST FOR ADDITION
            MOV      @R1,A             ;SAVE IN BUFFER
            MOV      A,#4              ;INDEX FOR CONTINUATION OF PROGRAM
            JMP      @A + DPTR         ;JUMP BACK INTO MAIN PROGRAM
```

This example illustrates several points:

1. The "subroutine" does not end with a normal return statement; instead, an indirect jump relative to the data pointer returns execution to the first instruction following the parameter list. The two initial POP instructions correct the stack-pointer contents.

2. Either an ACALL or LCALL works with the subroutine, since each pushes the address of the *next* instruction or data byte onto the stack. The call may be made from anywhere in the full 8051 address space, since the MOVC instruction accesses all 64K bytes.

3. The parameters passed to the utility can be listed in whatever order is most convenient, which may not be that in which they're used. The utility has essentially "random access" to the parameter list, by loading the appropriate constant into the accumulator before each MOVC instruction.

4. Other than the data pointer, the whole calling and processing sequence only affects the accumulator, PSW and pointer registers. The utility could have pushed these registers onto the stack (after popping the parameter list starting address), and popped before returning.

Passing parameters through in-line-code can be used in conjunction with other variable passing techniques.

The utility can also get input variables from working registers or from the stack, and return output variables to registers or to the stack.

PERIPHERAL INTERFACING TECHNIQUES

I/O Port Reconfiguration (First Approach)

I/O ports must often transmit or receive parallel data in formats other than as 8-bit bytes. For example, if an application requires three 5-bit latched output ports (called X, Y, and Z), these "virtual" ports could be mapped onto the pins of "physical" ports 1 and 2 (see example at bottom of page).

This pin assignment leaves P2.7 free for use as a test pin, input data pin, or control output through software.

Notice that the bits of port Z are reversed. The highest-order port Z pin corresponds to pin P2.2, and the lowest-order pin of port Z is P2.6, due to PC board layout considerations. When connecting an 8051 to an immediately adjacent keyboard column decoder or another device with weighted inputs, the corresponding pins may not be aligned. The interconnections must be "scrambled" to compensate either with interwoven circuit board traces or through software (as shown below and on the following page).

	PORT "Z"					PORT "Y"					PORT "X"				
-	PZ0	PZ1	PZ2	PZ3	PZ4	PY4	PY3	PY2	PY1	PY0	PX4	PX3	PX2	PX1	PX0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

```
        PX_MAP     DATA    20H
        PY_MAP     DATA    21H
        PZ_MAP     DATA    22H
        ;
        OUT_PX:    ANL     A,#00011111B    ;CLEAR BITS ACC.7 - ACC.5
                   MOV     PX_MAP,A        ;SAVE DATA IN MAP BYTE
                   ACALL   OUT_P1          ;UPDATE PORT 1 OUTPUT LATCH
                   RET

        ;          ...     ......
        OUT_PY:    MOV     PY_MAP,A        ;SAVE IN MAP BYTE
                   ACALL   OUT_P1          ;UPDATE PORT 1
                   ACALL   OUT_P2          ;AND PORT 2 OUTPUT LATCHES
                   RET

        ;          ...     ......
        OUT_PZ:    MOV     PZ_MAP,A        ;SAVE DATA IN MAP BYTE
                   ACALL   OUT_P2          ;UPDATE PORT 2.
                   RET

        ;          ...     ......
        ;          ...     ......
```

```
        OUT_P1:   MOV     A,PY_MAP           ;OUTPUT ALL P1 BITS
                  SWAP    A
                  RL      A                  ;SHIFT PY_MAP LEFT 5 BITS
                  ANL     A,#11100000B       ;MASK OUT GARBAGE
                  ORL     A,PX_MAP           ;INCLUDE PX_MAP BITS
                  MOV     P1,A
                  RET
        ;
        OUT_P2:   MOV     C,PZ_MAP.0         ;LOAD CY WITH P2.6 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  MOV     C,PZ_MAP.1         ;LOAD CY WITH P2.5 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  MOV     C,PZ_MAP.2         ;LOAD CY WITH P2.4 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  MOV     C,PZ_MAP.3         ;LOAD CY WITH P2.3 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  MOV     C,PZ_MAP.4         ;LOAD CY WITH P2.2 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  MOV     C,PZ_MAP.4         ;LOAD CY WITH P2.1 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  MOV     C,PZ_MAP.3         ;LOAD CY WITH P2.0 BIT
                  RLC     A                  ;AND SHIFT INTO ACC.
                  SETB    ACC.7              ;(ASSUMING INPUT ON P2.7)
                  MOV     P2.A
                  RET
```

Writing to the virtual ports must not affect any other pins. Since the virtual output algorithms are non-trivial, a subroutine is needed for each port: OUT_PX, OUT_PY and OUT_PZ. Each is called with data to output right-justified in the accumulator, and any data in bits ACC.7-ACC.5 is insignificant. Each subroutine saves the data in a "map" variable for the virtual port, then calls other subroutines which use the data in the various map bytes to compute and output the 8-bit pattern needed for each physical port affected. The two level structure of the above subroutines can be modified somewhat if code efficiency and execution speed are critical: incorporate the code shown as subroutines OUT_P1 and OUT_P2 directly into the code for OUT_PX and OUT_PZ, in place of the corresponding CALL instructions. OUT_PY would not be changed, but now the destinations for its ACALL instructions would be alternate entry points in OUT_PX and OUT_PZ, instead of isolated subroutines.

I/O Port Reconfiguration (Second Approach)

A trickier situation arises if two sections of code which write to the same port or register, or call virtual output routines like those above, need to be executed at different interrupt levels. For example, suppose the background program wants to rewrite Port X (using the port associations in the previous example), and has computed the bit pattern needed for P1. An interrupt is detected just before the MOV P1,A instruction, and the service routine tries to write Port Y. The service routine would correctly update P1 and P2, but upon returning to the background program P1 is immediately *re*-written with the data computed *before* the interrupt! Now pins P2.1 and P2.0 indicate (correctly) data written to port Y in the interrupt routine, but the earlier data written to P.7-P1.5 is no longer valid. The same sort of confusion could arise if a high-level interrupt disrupted such an output sequence.

One solution is to disable interrupts around any section of code which must not be interrupted (called a "critical section"), but this would adversely affect interrupt latency. Another is to have interrupt routines set or clear a flag ("semaphore") when a common resource is altered — a rather complex and elaborate system.

An easier way to ensure that any instruction which writes the port X field of P1 does not change the port Y field pins from their state *at the beginning of that instruction,* is shown next. A number of 8051 operations read, modify, and write the output port latches all in one instruction. These are the arithmetic and logical instructions (INC, DEC, ANL, ORL, etc.), where an addressed byte is both the destination variable and one of the source operands. Using these instructions, instead of data moves, eliminates the critical section problem entirely.

```
OUT_PX:     ANL     P1,#11100000B       ;CLEAR BITS P1.4-P1.0
            ORL     P1,A                ;SET P1 PIN FOR EACH ACC BIT SET
            RET
;           ...     ......
;           ...     ......
OUT_PY:     MOV     B,#20H
            MUL     AB                  ;SHIFT B A LEFT 5 BITS
            ANL     P1,#00011111B       ;CLEAR PY FIELD OF PORT 1
            ORL     P1,A                ;SET PY BITS ON PORT 1
            MOV     A,B                 ;'LOAD 2 BITS SHIFTED INTO B
            ANL     P2,#1111100B        ;AND UPDATE P2
            ORL     P2,A
            RET
;           ...     ......
;           ...     ......
OUT_PZ:     RRC     A                   ;MOVE ORIGINAL ACC.0 INTO CY
            MOV     P2.6,C              ;AND STORE TO PIN P2.6.
            RRC     A                   ;MOVE ORIGINAL ACC.1 INTO CY
            MOV     P2.5,C              ;AND STORE TO PIN P2.5.
            RRC     A                   ;MOVE ORIGINAL ACC.2 INTO CY
            MOV     P2.4,C              ;AND STORE TO PIN P2.4.
            RRC     A                   ;MOVE ORIGINAL ACC.3 INTO CY
            MOV     P2.3,C              ;AND STORE TO PIN P2.3.
            RRC     A                   ;MOVE ORIGINAL ACC.4 INTO CY
            MOV     P2.2,C              ;AND STORE TO PIN P2.2.
            RET
```

Simulating a Third Priority Level in Software

Some applications require more than the two priority levels that are provided by on-chip hardware in 8051 devices. In these cases, relatively simple software can be written to produce the same effect as a third priority level.

First, interrupts that are to have higher priority than 1 are assigned to priority 1 in the IP (Interrupt Priority) register. The service routines for priority 1 interrupts that are supposed to be interruptible by "priority 2" interrupts are written to include the following code:

```
            PUSH    IE
            MOV     IE,#MASK
            CALL    LABEL
;           ...     .....
;           (execute service routine)
;           ...     .....
            POP     IE
            RET
LABEL:      RETI
```

As soon as any priority 1 interrupt is acknowledged, the IE (Interrupt Enable) register is re-defined as as to disable all but "priority 2" interrupts. Then, a CALL to LABEL executes the RETI instruction, which clears the priority 1

interrupt-in-progress flip-flop. At this point any priority 1 interrupt that is enabled can be serviced, but only "priority 2" interrupts are enabled.

POPping IE restores the original enable byte. Then a normal RET (rather than another RETI) is used to terminate the service routine. The additional software adds 10 µs (at 12 MHz) to priority 1 interrupts.

Software Delay Timing

Many 8051 applications invoke exact control over output timing, A software-generated output strobe, for instance, might have to be *exactly* 50 µs wide. The DJNZ operation can insert a one instruction software delay into a piece of code, adding a moderate time delay of two instruction cycles per iteration. For example, two instructions can add a 49-µsec. software delay loop to code to generate a pulse on the WR pin.

```
            CLR     WR
            MOV     R2,#24
            DJNZ    R2,$
            SETB    WR
```

The dollar sign in this example is a special character meaning "the address of this instruction". It can be used to eliminate instruction labels on nearby source lines.

Serial Port and Timer Mode Configuration

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

To choose one arbitrary example, assume the 8051 should communicate with a standard CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. The resulting character rate is 2400 baud/9 bits, approximately 265 characters per second.

For the sake of clarity, the transmit and receive subroutines here are driven by simple-minded software status polling code rather than interrupts. The serial port must be initialized to 8-bit UART mode (SM0, SM1 = 01), enabled to receive all messages (SM2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. All this can be set up with the instruction at label SPINIT.

Timer 1 will be used in auto-reload mode as a baud rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by

$$\frac{1 \times 10^6}{(32)\,(2400)}$$

which equals 13 (actually, 13.02) instruction cycles. The timer must reload the value 13, or 0F3H, as shown by the code at label TIINIT. (ASM51 will accept both the signed decimal or hexadecimal representations.)

```
;              INITIALIZE SERIAL PORT
;              FOR 8-BIT UART MODE
;              & SET TRANSMIT READY FLAG.
SPINIT:    MOV     SCON,#01010010B
;
;              INITIALIZE TIMER 1 FOR
;              AUTO-RELOAD AT 32 X 2400 HZ
;              (TO USED AS GATED 16-BIT COUNTER.)
;
TIINIT:    MOV     TCON,#11010010B
           MOV     TH1,#13
           SETB    TR1
;          ...     ......
;
```

Simple Serial I/O Drivers

SP_OUT is a simple subroutine to transmit the character passed to it in the accumulator. First it must compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and then return.

SP_IN is an equally simple routine which waits until a character is received, sets the carry flag if there is an odd-parity error, and returns the masked seven-bit code in the accumulator.

```
;
;SP_OUT     ADD ODD PARITY TO ACC AND
;           TRANSMIT WHEN SERIAL PORT READY
;
SP_OUT:    MOV     C,P            ;MOVE PARITY BIT TO CARRY BIT
           CPL     C
           MOV     ACC.7,C        ;INSERT INTO DATA BYTE
           JNB     TI,$           ;WAIT FOR TRANSMITTER AVAILABLE
           CLR     TI
           MOV     SBUF,A         ;OUTPUT THE CHARACTER
           RET
;
;
```

```
SP_IN:      JNB     RI,$            ;WAIT FOR A CHARACTER TO BE RECEIVED
            CLR     RI
            MOV     A,SBUF          ;MOVE CHARACTER TO THE ACCUMULATOR
            MOV     C,P
            CPL     C               ;SET CARRY BIT TO ONE IF ODD-PARITY ERROR
            ANL     A,#7FH          ;MASK OUT PARITY BIT FROM CHARACTER
            RET
```

Transmitting Serial Port Character Strings

Any application which transmits characters through a serial port to an ASCII output device will on occasion need to output "canned" messages, including error messages, diagnostics, or operator instructions. These character strings are most easily defined with in-line data bytes defined with the DB directive.

```
CR          EQU     0DH             ;ASCII CARRIAGE RET
LF          EQU     0AH             ;ASCII LINE-FEED
ESC         EQU     1BH             ;ASCII ESCAPE CODE
;           ...     ......
            CALL    XSTRING
            DB      CR,LF           ;NEW LINES
            DB      'AMD QUALITY'   ;MESSAGE
            DB      ESC             ;ESCAPE CHARACTER
;
;           (CONTINUATION OF PROGRAM)
;
;           ...     ......
XSTRING:    POP     DPH             ;LOAD DPTR WITH FIRST CHARACTER
            POP     DPL
XSTR_1:     CLR     A               ;(ZERO OFFSET)
            MOVC    A,@A + DPTR     ;FETCH FIRST CHARACTER OF STRING
XSTR_2:     JNB     TI,$            ;WAIT UNTIL TRANSMITTER READY
            CLR     TI              ;MARK AS NOT READY
            MOV     SBUF,A          ;OUTPUT NEXT CHARACTER
            INC     DPTR            ;BUMP POINTER
            CLR     A
MOVC        A,@A + DPTR             ;GET NEXT OUTPUT CHARACTER
            CJNE    A,#ESC,XSTR_2   ;LOOP UNTIL ESCAPE READ
            MOV     A,#1
            JMP     @A + DPTR       ;RETURN TO CODE AFTER ESCAPE
```

Recognizing and Processing Special Cases

Before operating on the data it receives, a subroutine might give "special handling" to certain input values. Consider a word processing device which receives ASCII characters through the 8051 serial port and drives a thermal hard-copy printer. A standard routine translates most printing characters to bit patterns, but certain control characters (<DEL>, <CR>, <LF>, <BEL>, <ESC>, or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters. The CJNE operation provides essentially a one-instruction CASE statement.

```
;
CHAR        EQU     R7              ;CHARACTER CODE VARIABLE
;
INTERP:     CJNE    CHAR,#7FH,INTP_1 ;SKIP UNLESS RUBOUT
;           ...     ......          (SPECIAL ROUTINE FOR RUBOUT CODE)
            RET
INTP_1:     CJNE    CHAR,#07H,INTP_2 ;SKIP UNLESS BELL
;           ...     ......          (SPECIAL ROUTINE FOR BELL CODE)
            RET
```

```
INTP_2:     CJNE    CHAR,#0AH,INTP_3    ;SKIP  UNLESS  LFEED
;           ...     ......              (SPECIAL  ROUTINE  FOR  LFEED  CODE)
            RET
INTP_3:     CJNE    CHAR,#0DH,INTP_4    ;SKIP  UNLESS  RETURN
;           ...     ......              (SPECIAL  ROUTINE  FOR  RETURN  CODE
            RET
INTP_4:     CJNE    CHAR,#1BH,INTP_5    ;SKIP  UNLESS  ESCAPE
;           ...     ......              (SPECIAL  ROUTINE  FOR  ESCAPE  CODE)
            RET
INTP_5:     CJNE    CHAR,#20H,INTP_6    ;SKIP  UNLESS  SPACE
;           ...     ......              (SPECIAL  ROUTINE  FOR  SPACE  CODE)
            RET
INTP_6:     JC      PRINTC             ;JUMP  IF  CODE  20H
            MOV     CHAR,#0            ;REPLACE  CONTROL  CHARACTER  WITH
                                       ;NULL  CODE
PRINTC:                                ;PROCESS  STANDARD  PRINTING
;           ...     ......             ;CHARACTER
            RET
;
```

Buffering Serial Port Output Characters

It is not always efficient to transmit characters through the serial port one-at-a-time. Most applications generate a short burst of characters all at once (English words or multi-digit numbers, for instance), with the bursts themselves occurring at longer intervals. Instead of waiting while the UART outputs each character, it would be more efficient if the background program could enter all the characters into a first-in first-out (FIFO) data structure, and continue about its business, letting an interrupt routine transmit each character as the serial port becomes available.

Assume there is a 16-byte output data buffer starting at 70H. QHEAD and QTAIL keep track of the head and tail portion of the buffer being used. The subroutine ENTERQ waits until there is space in the queue, then copies a character code from the accumulator to the queue.

```
QHEAD       DATA    6EH                ;LAST BYTE ENTERED INTO QUEUE
QTAIL       DATA    6FH                ;LAST BYTE READ FROM QUEUE
BOTLIM      EQU     70H
TOPLIM      EQU     7FH
;
;           QUEUE IS EMPTY WHEN QHEAD = QTAIL AND
;           FULL WHEN Q HEAD + 1 (WITHIN RANGE) = QTAIL.
            MOV     QHEAD,#TOPLIM
            MOV     QTAIL,#TOPLIM
;
;
ENTERQ:     MOV     R0,A               ;SAVE ACC DATA
            MOV     A,QHEAD            ;LOAD HEAD POINTER
            INC     A                  ;PRE-INCREMENT POINTER
            CJNE    A,#TOPLIM+1,ENTQ_1
            MOV     A,#BOTLIM          ;RELOAD ON OVERFLOW
ENTQ_1:     CJNE    A,QTAIL,ENTQ_2     ;TEST IF QUEUE FULL
            SJMP    ENTQ_1             ;LOOP UNTIL SPACE AVAILABLE
ENTQ_2:     XCH     A,R0               ;STORE POINTER AND RELOAD ACC
            MOV     @R0,A              ;ENTER INTO QUEUE
            MOV     QHEAD,R0           ;UPDATE HEAD POINTER
            SETB    ES                 ;ENABLE SERIAL PORT INTERRUPTS
            RET
```

The interrupt routine DQUEUE is invoked when the transmitter is ready for another character. First it determines if any characters are available for transmission, indicated by QHEAD and QTAIL being not equal. If more data is available, it is written to the transmit buffer (SBUF) and the pointers are updated. If not, DQUEUE disables serial port interrupts and returns to the background program. ENTERQ will re-enable such interrupts as more data is available. (This example does not consider interrupt-driven serial input.)

```
                ORG     0023H
                PUSH    ACC                     ;SAVE  CPU  STATUS
                PUSH    PSW
                MOV     PSW,#30Q                ;SELECT  BANK  3
DQUEUE:         MOV     A,QTAIL
                CJNE    A,QHEAD,DQ_1            ;TEST  IF  QUEUE  EMPTY
                CLR     ES                      ;IF  SO,  CLEAR  ENABLE  BIT  AND  RETURN
                SJMP    TI_RET
DQ_1:           CLR     TI                      ;ELSE  ACKNOWLEDGE  REQUEST
                INC     A                       ;COMPUTE  NEXT  BYTE'S  ADDRESS
                CJNE    A,#TOPLIM+1,DQ_2
                MOV     A,#BOTLIM               ;REVISE  ACC  IF  POINTER  OVERFLOWED
DQ_2:           MOV     R0,A                    ;LOAD  INDEX  REGISTER
                MOV     SBUF,@R0                ;RELOAD  TRANSMITTER
                MOV     QTAIL,A                 ;SAVE  LAST  POINTER  USED.
TI_RET:         POP     PSW                     ;RESTORE  STATUS  AND  RETURN
                POP     ACC
                RETI
```

Synchronizing Timer Overflows

8051 timer overflows automatically generate an internal interrupt request, which will vector program execution to the appropriate interrupt service routine if interrupts are enabled and no other service routines are in progress at the time. However, it is not predictable *exactly* how long it will take to reach the service routine. The service routine call takes two instruction cycles, but 1, 2, or 4 additional cycles may be needed to complete the instruction in progress. If the background program ever disables interrupts, the response latency could further increase by a few instruction cycles. (Critical sections generally involve simple instruction sequences — rarely multiplies or divides.) Interrupt response delay is generally negligible, but certain time-critical applications must take the exact delay into account. For example, generating interrupts with timer 1 every millisecond (1000 instruction cycles) or so would normally call for reloading it with the value, -1000 (0FC18H). But if the interrupt interval (average over time) must be accurate to 1 instruction cycle, the 16-bit value reload into the timer must be computed, taking into account when the timer actually overflowed.

This simply requires reading the appropriate timer, which has been incremented each cycle since the overflow occurred. A sequence like the one below can stop the timer, computer how much time should elapse before the next interrupt, and reload and restart the timer. The double-precision calculation shown here compensates for any amount of timer overrun within the maximum interval. Note that it also takes into account that the timer is stopped for seven instruction cycles in the process. All interrupts are disabled, so a higher priority request will not be able to disrupt the time-critical code section.

```
;               ...     ......
                CLR     EA                      ;DISABLE  ALL  INTERRUPTS
                CLR     TR1                     ;STOP  TIMER  1
                MOV     A,#LOW(−1000+7)         ;LOAD  LOW-ORDER  DESIRED  COUNT
                ADD     A,TL1                   ;CORRECT  FOR  TIMER  OVERRUN
                MOV     TL1,A                   ;RELOAD  LOW-ORDER  BYTE
                MOV     A,#HIGH(−1000+7)        ;REPEAT  FOR  HIGH-ORDER  BYTE
                ADDC    A,TH1
                MOV     TH1,A
                SETB    TH1                     ;RESTART  TIMER
;               ...     ......
```

Reading a Timer/Counter "On-the-Fly"

The preceding example simply stopped the timer before changing its contents. This is normally done when reloading a timer so that the time at which the timer is started (i.e. the "run" flag is set) can be exactly controlled. There are situations, though, when it is desired to read the current count without disrupting the timing process. The 8051 timer/counter registers can all be read or written while they are running, but a few precautions must be taken.

Suppose the subroutine RDTIME should return in <R1> <R0> a 16-bit value indicating the count in timer 0. The instant at which the counter was sampled is not as critical as the fact that the value returned must have been valid at some point while the routine was in progress. There is a potential problem that between reading the two halves, a low-order register overflow might increment the high-order register, and the two data bytes returned would be "out of phase". The solution is to read the high-order byte first, then the low-order byte, and then confirm that the high-order byte has not changed. If it has, repeat the whole process.

```
RDTIME:    MOV    A,TH0          ;SAMPLE  TIMER0   (HIGH)
           MOV    R0,TL0         ;SAMPLE  TIMER0   (LOW)
           CJNE   A,TH0,RDTIME   ;REPEAT  IF  NECESSARY
           MOV    R1,A           ;STORE  VALID  READ
           RET
```

CHAPTER 6

8051 Family Boolean Processing Capabilities

The 8051 incorporates a number of special features that support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the 8051 Family *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities. The purpose of this discussion is to explain these concepts and show how they are used.

BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts that have been around for sometime. Digital computer systems of widely varying designs all have four functional elements in common (Figure 6-1):

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions,

- a memory to store the sequence of instructions making up a program or algorithm,

- data memory to store variables used by the program, and

- some means of communicating with the outside world.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at the minimum, operation classes to perform arithmetic or logical functions on program variables, to move variables from one place to another, to cause program execution to jump or conditionally branch based on register or variable states, and to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from its ability to combine simple, fast operations to produce more complex (albeit slower) ones, which in turn link together to eventually solve the problem at hand. A 4-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the 4-bit CPU can simulate a far more complex "virtual" machine.

In fact, any digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word
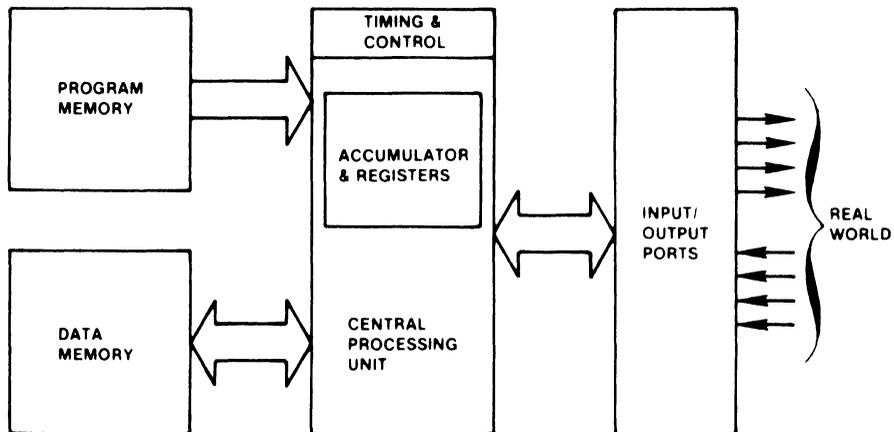
Figure 6-1. Block Diagram for Abstract Digital Computer

processors might first re-create Shakespeare's classics and this chapter)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given task is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital-control problems.

Processing Elements

The following shows how the four basic elements of a digital computer — a CPU with associated registers, program memory, addressable data RAM, and I/O capabilities — relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 6-1. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 6-1, several sophisticated program control features, like multiple addressing modes, subroutine nesting, and a two-level interrupt structure, are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 6-2). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 6-2). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 Family.

Table 6-1. 8051 Family Boolean Processing Instruction Subset

Mnemonic		Description	Byte	Cyc
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct bit	2	1
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
MOV	C,bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,bit	AND complement of direct bit to Carry flag	2	2
ORL	C,bit	OR direct bit to Carry flag	2	2
ORL	C,bit	OR complement of direct bit to Carry flag	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct bit set	3	2
JNB	bit,rel	Jump if direct bit not set	3	2
JBC	bit,rel	Jump if direct bit is set & Clear bit	3	2

Address mode abbreviations

C — Carry flag.
bit — 128 software flags, any I/O pin, control or status bit.
rel — All conditional jumps include an 8-bit offset byte. Range is +127 −128 bytes relative to first byte of the following instruction.

Data Memory. The instructions in Figure 6-2 can operate directly upon 144 general-purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register; in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU, and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).

```
┌─────────┐
│ opcode  │
└─────────┘
```

SETB C
CLR C
CPL C

```
┌─────────┐ ┌──────────────┐
│ opcode  │ │ displacement │
└─────────┘ └──────────────┘
```

JC   rel
JNC   rel

a. Carry Control and Test Instructions

```
┌─────────┐ ┌──────────────┐
│ opcode  │ │ bit address  │
└─────────┘ └──────────────┘
```

SETB   bit
CLR   bit
CPL   bit
ANL C,   bit
ANL C,/   bit
ORL C,   bit
ORL C,/   bit
MOV C,   bit
MOV   bit,C

```
┌─────────┐ ┌──────────────┐ ┌──────────────┐
│ opcode  │ │ bit address  │ │ displacement │
└─────────┘ └──────────────┘ └──────────────┘
```

JB   bit,   rel
JNB   bit,   rel
JBC   bit,   rel

b. Bit Manipulation and Test Instructions

Figure 6-2. Bit Addressing Instruction Formats

Direct Bit Addressing

The most significant bit of the direct-address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 16 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 6-3a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM; the five high-order address bits match those of the register's own

address, while the three low-order bits identify the bit position within that register (Figure 6-3b).

Notice the column labeled "Symbol" in Figure 6-4. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C. P0, P1, P2, and P3 are the 8051's four I/O ports; secondary functions assigned to each of the eight pins of P3 are shown in Figure 6-5.

Figure 6-6 shows the last four bit-addressable registers. TCON (Timer Control) and SCON (Serial-Port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed; they *cannot* be used as software flags.

Addressable Register Set. There are 20 special-function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are reserved for expansion in microcomputers based on the 8051 Family architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logic operations can be used to manipulate bits in all *non*-bit-addressable registers and RAM.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 6-4 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can, therefore, affect the carry.

Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 6-1 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition. SETB and CLR are analogous to loading a bit with a constant, 1 or 0. Single byte versions perform the same three operations on the carry.

a. Ram Bit Addresses

RAM Byte	(MSB)							(LSB)
7FH								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH 18H	Bank 3							
17H 10H	Bank 2							
0FH 08H	Bank 1							
07H 00	Bank 0							
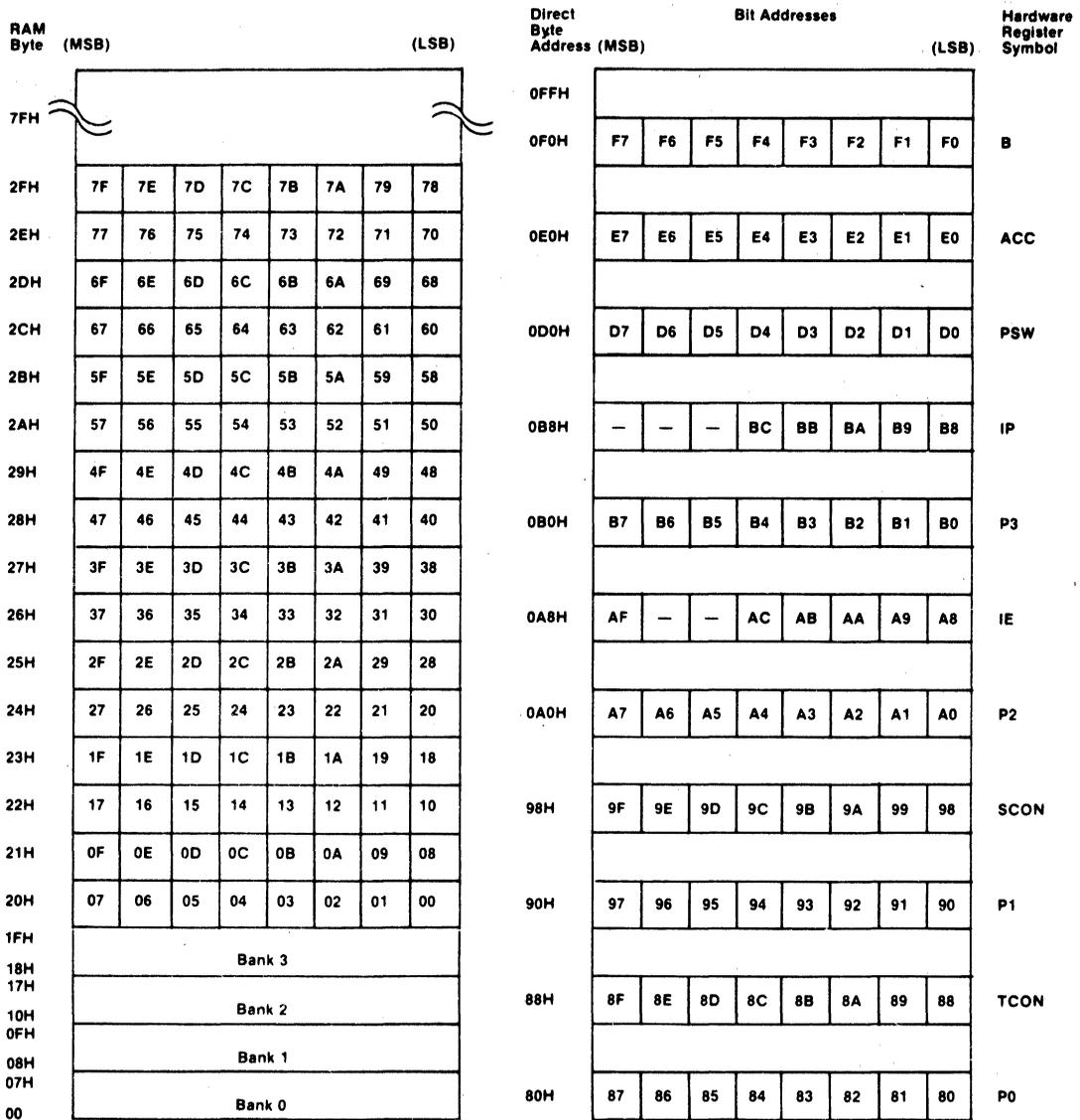
b. Special Function Register Bit Addresses

Direct Byte Address	(MSB)		Bit Addresses					(LSB)	Hardware Register Symbol
0FFH									
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0B8H	—	—	—	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A8H	AF	—	—	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
80H	87	86	85	84	83	82	81	80	P0

Figure 6-3. Bit Address Maps

(MSB)   **(LSB)**

CY	AC	F0	RS1	RS0	OV	—	P

Symbol Position Name and Significance

CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0. Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits.
RS0	PSW.3	1 & 0. Set/cleared by software to determine working register bank (see Note).
OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.
Note-		the contents of (RS1, RS0) enable the working register banks as follows: (0,0) - Bank 0 (00H–07H) (0,1) - Bank 1 (08H–0FH) (1,0) - Bank 2 (10H–17H) (1,1) - Bank 3 (18H–1FH)

Figure 6-4. PSW — Program Status Word Organization

(MSB)   **(LSB)**

RD	WR	T1	T0	INT1	INT0	TXD	RXD

Symbol Position Name and Significance

RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin.
T0	P3.4	Timer/counter 0 external input or test pin.
INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RXD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6-5. P3 — Alternate I/O Functions of Port 3

(MSB)   **(LSB)**

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol Position Name and Significance

TF1   TCON.7   Timer 1 overflow Flag.
                 Set by hardware on timer/
                 counter overflow. Cleared
                 when interrupt processed.

TR1   TCON.6   Timer 1 Run control bit.
                 Set/cleared by software to turn
                 timer/counter on/off.

TF0   TCON.5   Timer 0 overflow Flag.
                 Set by hardware on timer/
                 counter overflow. Cleared
                 when interrupt processed.

TR0   TCON.4   Timer 0 Run control bit.
                 Set/cleared by software to turn
                 timer/counter on/off.

IE1   TCON.3   Interrupt 1 Edge flag.
                 Set by hardware when exter-
                 nal interrupt edge detected.
                 Cleared when interrupt pro-
                 cessed.

IT1   TCON.2   Interrupt 1 Type control bit.
                 Set/cleared by software to
                 specify falling edge/low level
                 triggered external interrupts.

IE0   TCON.1   Interrupt 0 Edge flag.
                 Set by hardware when exter-
                 nal interrupt edge detected.
                 Cleared when interrupt pro-
                 cessed.

IT0   TCON.0   Interrupt 0 Type control bit.
                 Set/cleared by software to
                 specify falling edge/low level
                 triggered external interrupts.

a. TCON—Timer/Counter Control/Status Register

(MSB)   **(LSB)**

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol Position Name and Significance

SM0   SCON.7   Serial port Mode control bit 0.
                 Set/cleared by software (see
                 note).

SM1   SCON.6   Serial port Mode control bit 1.
                 Set/cleared by software (see
                 note).

SM2   SCON.5   Serial port Mode control bit 2.
                 Set by software to disable re-
                 ception of frames for which bit
                 8 is zero.

REN   SCON.4   Receiver Enable control bit.
                 Set/cleared by software to en-
                 able/disable serial data recep-
                 tion.

TB8   SCON.3   Transmit Bit 8.
                 Set/cleared by hardware to de-
                 termine state of ninth data bit
                 transmitted in 9-bit UART
                 mode.

RB8   SCON.2   Receive Bit 8.
                 Set/cleared by hardware to in-
                 dicate state of ninth data bit
                 received.

TI   SCON.1   Transmit Interrupt flag.
                 Set by hardware when byte
                 transmitted. Cleared by soft-
                 ware after servicing.

RI   SCON.0   Receive Interrupt flag.
                 Set by hardware when byte re-
                 ceived. Cleared by software
                 after servicing.

Note-   the state of (SM0, SM1)
            selects:
            (0,0)—Shift register I/O
                    expansion.
            (0,1)—8-bit UART, variable
                    data rate.
            (1,0)—9-bit UART, fixed data
                    rate.
            (1,1)—9-bit UART, variable
                    data rate.

b. SCON—Serial Port Control/Status Register

Figure 6-6. Peripheral Configuration Registers

(MSB)   (LSB)

EA	—	—	ES	ET1	EX1	ET1	EX0

Symbol Position Name and Significance

EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4–IE.0.
—	IE.6	(reserved)
—	IE.5	
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.

EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

c. IE—Interrupt Enable Register

(MSB)   (LSB)

—	—	—	PS	PT1	PX1	PT0	PX0

Symbol Position Name and Significance

—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

d. IP—Interrupt Priority Contol Register

Figure 6-6. Peripheral Configuration Registers (continued)

ASM51 specifies a bit address in any of three ways:

- by the number or expression corresponding to the direct bit address (0-255);

- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0);

- in the case of control and status register, by the predefined assembler symbols listed in the first columns of Figures 6-4 through 6-6.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

```
USR_FLG BIT  PSW.5     ; User Symbol Definition
;    ...     .....
     CLR     0D5H      ; Absolute Addressing
     CLR     PSW.5     ; Use of Dot Operator
     CLR     F0        ; Pre-Defined Assembler Symbol
     CLR     USR_FLG   ; User-Defined Symbol
```

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via a carry by combining the two instructions. (If necessary, one may push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 6-7, which shows a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. The letters "rel" denote relative code addressing. The 3-byte instructions "JB bit, rel" and "JNB bit, rel" (Jump on Bit and Jump on Not Bit) test the state of *any* addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit, rel" conditionally branches to the indicated address, then clears the bit in the same 2-cycle instruction. This operation is the same as the 8048-family "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from −128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a programmer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed, giving only the destination address or label. An error message will alert the programmer if the destination is "out of range."

The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logic-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result.
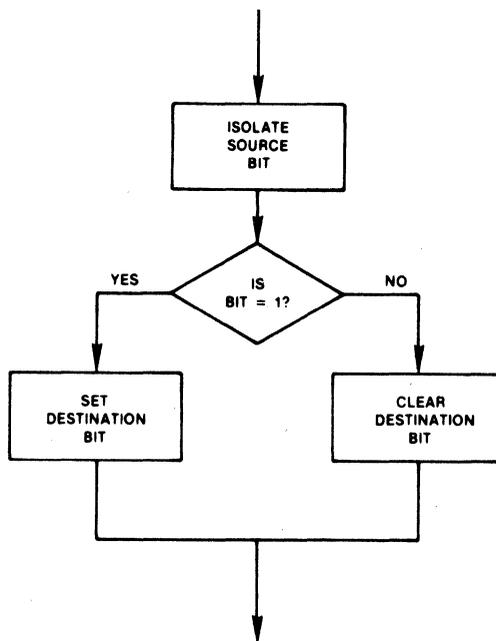

Figure 6-7. Bit Transfer Instruction Operation

This is essentially equivalent to the 8051 "MOV C,bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the 8051 Family sense. A flag exists only as a field within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.

Interaction with Other Instructions. The carry flag is also affected by the instructions listed in Table 6-2. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer-counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flag (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. These bits are all part of the TCON register, Figure 6-6a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function, and does not need to memorize control word layouts.

In the 8048-family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear and complement the software flags F0 and F1, enable and disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB, CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the 8048 software flags, the I/O pins, T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 6-2. Other Instructions Affecting the Carry Flag

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC	A,direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC	A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB	A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1
RLC	A	Rotate Accumulator Left through the Carry flag	1	1
RRC	A	Rotate Accumulator Right through Carry flag	1	1
CJNE	A,direct,rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE	A,#data,rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Compare immed to register & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Compare immed to indirect & Jump if Not Equal	3	2

Table 6-3a shows how 8051 programs implement software flag and machine control functions associated with special opcodes in the 8048. In every case the 8051 Family solution requires the same number of machine cycles, and executes 2.5 times faster.

Table 6-3a. Contrasting 8048 and 8051 Bit Control and Testing Instructions

8048 Instruction		Bytes	Cycles	μs	8051 Instruction		Bytes	Cycles & μs
Flag Control								
CLR	C	1	1	2.5	CLR	C	1	1
CPL	F0	1	1	2.5	CPL	F0	2	1
Flag Testing								
JNC	offset	2	2	5.0	JNC	rel	2	2
JF0	offset	2	2	5.0	JB	F0,rel	3	2
JB7	offset	2	2	5.0	JB	ACC.7,rel	3	2
Peripheral Polling								
JT0	offset	2	2	5.0	JB	T0,rel	3	2
JN1	offset	2	2	5.0	JNB	INT0,rel	3	2
JTF	offset	2	2	5.0	JBC	TF0,rel	3	2
Machine and Peripheral Control								
STRT	T	1	1	2.5	SETB	TR0	2	1
EN	1	1	1	2.5	SETB	EX0	2	1
DIS	TCNT1	1	1	2.5	CLR	ET0	2	1

Table 6-3b. Replacing 8048 Instruction Sequences with Single 8051 Instructions

8048 Instruction		Bytes	Cycles	μs	8051 Instruction		Bytes	Cycles & μs
Flag Control								
Set carry								
CLR	C				SETB	C	1	1
CPL	C	= 2	2	5.0				
Set Software Flag								
CLR	F0				SETB	F0	2	1
CPL	F0	= 2	2	5.0				
Turn Off Output Pin								
ANL	P1, #0FBH	= 2	2	5.0	CLR	P1.2	2	1
Complement Output Pin								
IN	A,P1							
XRL	A, #04H							
OUTL	P1,A	= 4	6	15.0	CPL	P1.2	2	1
Clear Flag in RAM								
MOV	R0, # FLGADR							
MOV	A,@R0							
ANL	A, # FLGMASK							
MOV	@R0,A	= 6	6	15.0	CLR	USER__FLG	2	1

Table 6-3b. Replacing 8048 Instruction Sequences with Single 8051 Instructions (continued)

8048 Instruction		Bytes	Cycles	μs	8051 Instruction		Bytes	Cycles & μs
Flag Testing:								
Jump if Software Flag is 0								
JF0	$ + 4							
JMP	offset = 4		4	10.0	JNB	F0,rel	3	2
Jump if Accumulator bit is 0								
CPL	A							
JB7	offset							
CPL	A = 4		4	10.0	JNB	ACC.7,rel	3	2
Peripheral Polling								
Test if Input Pin is Grounded								
IN	A.P1							
CPL	A							
JB3	offset = 4		5	12.5	JNB	P1.3,rel	3	2
Test if Interrupt Pin is High								
JN1	$ + 4							
JMP	offset = 4		4	10.0	JB	INT0,rel	3	2

BOOLEAN PROCESSOR APPLICATIONS

So what does all this buy you?

Qualitatively, nothing. All the same capabilities *could* be (and often have been) implemented on other machines using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

Quantitatively, the differences between a solution provided by the 8051 and those required by previous architectures are numerous. The 8051 Family solution is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048 instructions in a few general operations has been used to add new functionality to the 8051 family architecture — both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 6-3b. Here the 8051 speed advantage ranges from 5x to 15x!

Combining Boolean and byte-wide instructions can produce great synergy. An 8051 Family based application will prove to be:

• simpler to write since the architecture correlates more closely with the problems being solved;

• easier to debug because more individual instructions have no unexpected or undesirable side-effects;

• more byte efficient due to direct bit addressing and program counter relative branching;

• faster running because fewer bytes of instructions need to be fetched and fewer conditional jumps are processed;

• lower cost because of the high level of system-integration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

Design Example #1 — Bit Permutation

First, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

It is not the purpose here to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive-OR operations, function table look-ups, and an extensive sequence of bit permutation, packing, and unpacking steps. The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 6-8. This step is repeated 16 times for each key used in the course of a transmission. In essence, a 7-byte, 56-bit "Shift Key Buffer" is transformed into an 8-byte, "Permutation Buffer" without altering the shifted key. The arrows in Figure 6-8 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used;

the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 6-9a:

- Initialize the Permutation Buffer to default state (ones or zeroes);

- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte;

- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct;

- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 μs on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 6-7. The Boolean Processor can permute bits by simply moving them from the source to the carry to the destination — a total of two instructions taking 4 bytes and 3 μs per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB_1, SKB_2 . . . SKB_56. Then working from Figure 6-8, the software for the permutation algorithm would be that of Example 6-1a. The total routine length would be 192 bytes, requiring 144 μs.

Permuted and Shifted 56-Bit Key Buffer

48-Bit Key K_i
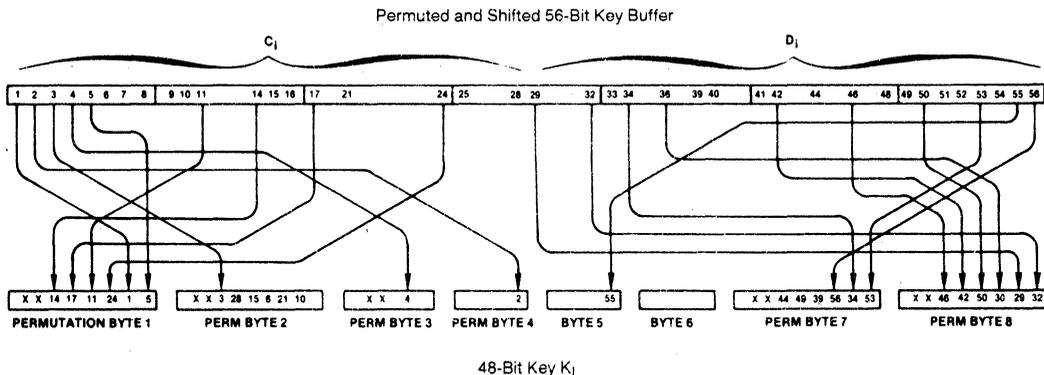
Figure 6-8. DES Key Schedule Transformation
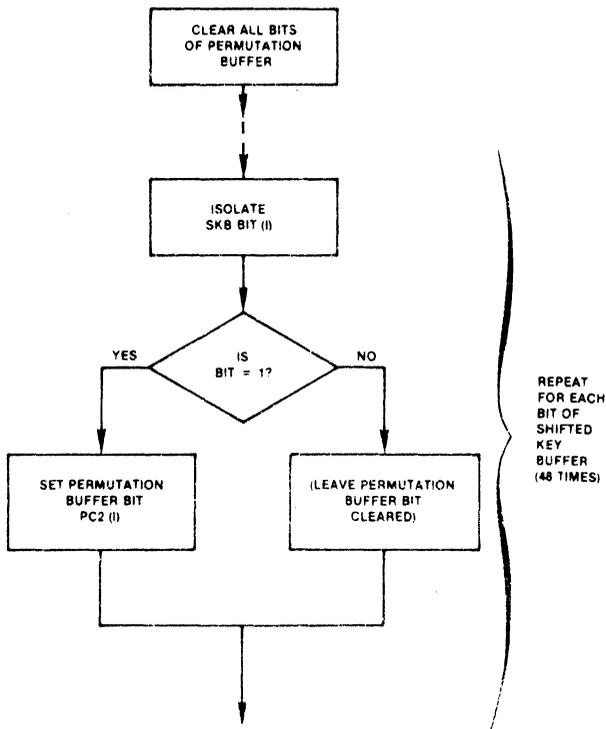
Figure 6-9a. Flowchart for Key Permutation Attempted with a Byte Processor

The algorithm of Figure 6-9b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 μs.) and shifting it into the accumulator (1 μs.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 6-1b. Worst-case execution time would be 112 μs, since each instruction takes a single cycle. Routine length should also decrease, to 168 bytes. Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated — saving memory and cutting execution time by another 8 μs.

```
                    ┌─────────────────────────────┐
                    │     CLEAR ACCUMULATOR        │
                    └─────────────────────────────┘

            ┌─────────────────────────────────────────┐
            │   LOAD BIT MAPPED ONTO BIT 5 OF          │
            │   PERMUTATION BYTE INTO CARRY            │
            └─────────────────────────────────────────┘

                ┌─────────────────────────────────┐
                │    ROTATE LEFT INTO ACC.         │
                └─────────────────────────────────┘

            ┌─────────────────────────────────────────┐
            │   LOAD BIT MAPPED ONTO BIT 4             │
            │   OF PERMUTATION BYTE INTO CARRY         │
            └─────────────────────────────────────────┘

                ┌─────────────────────────────────┐
                │    ROTATE LEFT INTO ACC.         │
                └─────────────────────────────────┘

            ┌─────────────────────────────────────────┐
            │   LOAD BIT MAPPED ONTO BIT 0             │
            │   OF PERMUTATION BYTE INTO CARRY         │
            └─────────────────────────────────────────┘

                ┌─────────────────────────────────┐
                │    ROTATE LEFT INTO ACC.         │
                └─────────────────────────────────┘

            ┌─────────────────────────────────────────┐
            │   STORE ACC. INTO PERMUTATION            │
            │   BUFFER                                 │
            └─────────────────────────────────────────┘
```
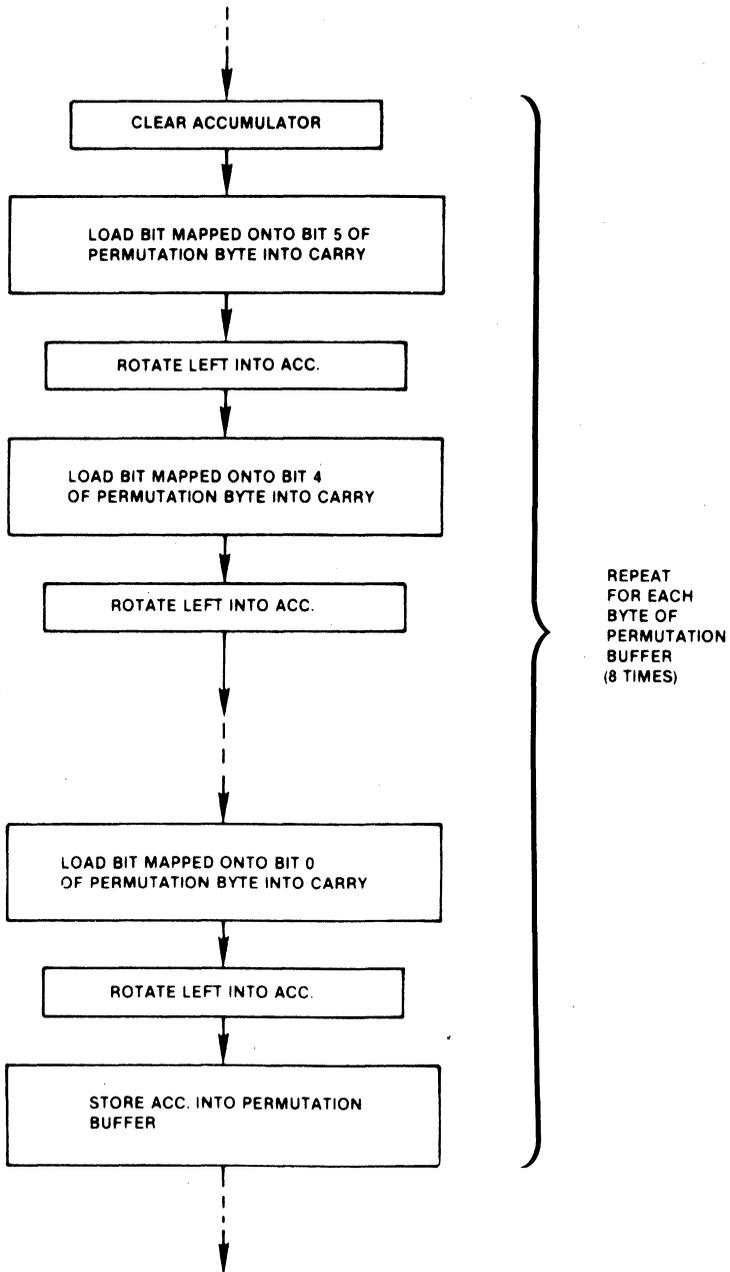
REPEAT
FOR EACH
BYTE OF
PERMUTATION
BUFFER
(8 TIMES)

Figure 6-9b. DES Key Permutation with Boolean Processor

Example 6-1. DES Key Permutation Software

a. "Brute Force" technique

MOV	C,SKB_1
MOV	PB_1.1,C
MOV	C,SKB_2
MOV	PB_4.0,C
MOV	C,SKB_3
MOV	PB_2.5,C
MOV	C,SKB_4
MOV	PB_1.0,C
. . .	. . . . .
. . .	. . . . .
MOV	C,SKB_55
MOV	PB_5.0,C
MOV	C,SKB_56
MOV	PB_7.2,C

b. Using Accumulator to Collect Bits

CLR	A
MOV	C,SKB_14
RLC	A
MOV	C,SKB_17
RLC	A
MOV	C,SKB_11
RLC	A
MOV	C,SKB_24
RLC	A
MOV	C,SKB_1
RLC	A
MOV	C,SKB_5
RLC	A
MOV	PB_1,A
. . .	. . . . .
. . .	. . . . .
MOV	C,SKB_29
RLC	A
MOV	C,SKB_32
RLC	A
MOV	PB_8,A

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption decryption process, and still more hardware to form a serial bit stream for transmission (Figure 6-10a). An 8051 solution could pack most of the entire system onto the one chip (Figure 6-10b). The whole DES algorithm would require less than one-fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

Design Example #2 — Software Serial I/O

An example often imposed on beginning microcomputer students is to write a program simulating a UART. Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product-line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figure 6-11a and 11b, show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

A side-by-side comparison of the software for this common application with three different microprocessor architectures is shown in Table 6-4a and 6-4b. The 8051 solution is more efficient than the others on every count!
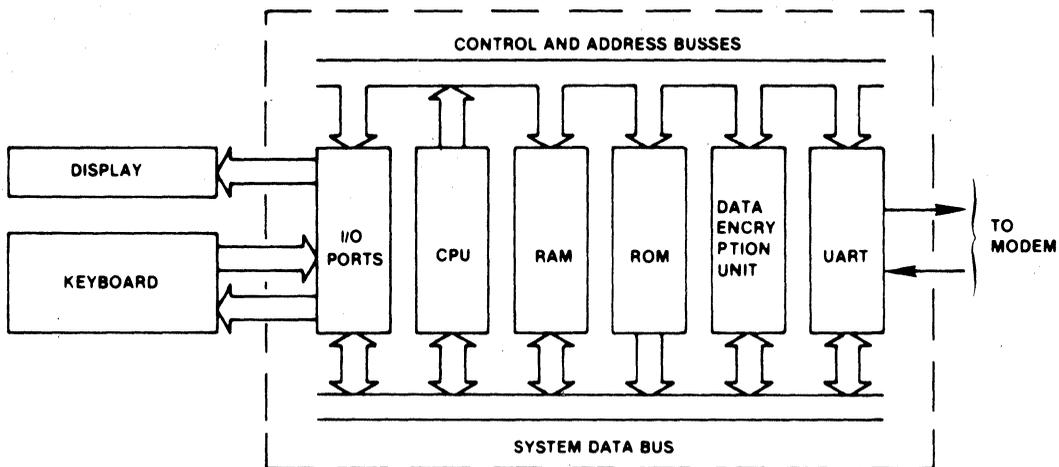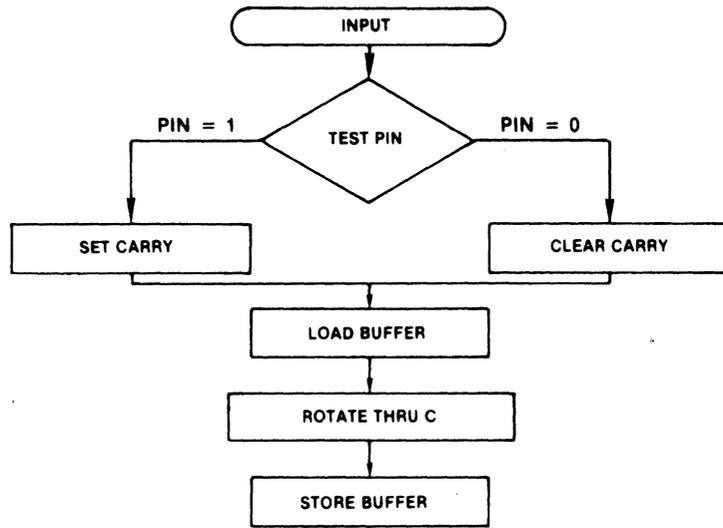
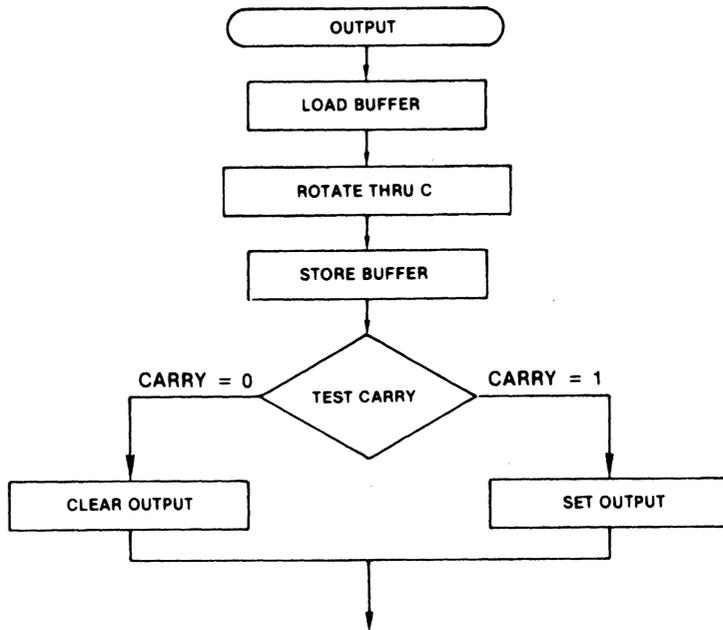a. Using Multi-Chip Processor Technology
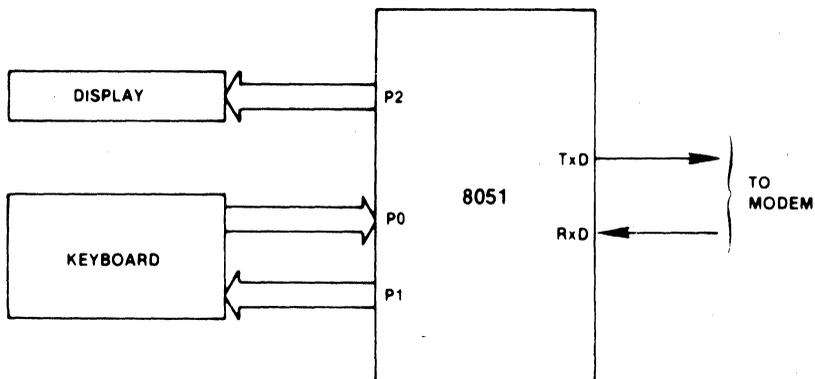
b. Using One Single-Chip Microcomputer

Figure 6-10. Secure Banking Terminal Block Diagram

a. Reception

b. Transmission

Figure 6-11. Serial I/O Algorithms

Table 6-4. Serial I/O Programs for Various Microprocessors

a.) Input Routine

8085		8048		8051	
IN	SERPORT			MOV	C,SERPIN
ANI	MASK	CLR	C		
JZ	LO	JNTO	LO		
CMC		CPL	C		
LO: LXI	HL,SERBUF	MOV	RO,#SERBUF		
MOV	A,M	MOV	A,@RO	MOV	A,SERBUF
RR		RRC	A	RRC	A
MOV	M,A	MOV	@RO,A	MOV	SERBUF,A

RESULTS:

8 Instructions	7 Instructions	4 Instructions
14 Bytes	9 Bytes	7 Bytes
56 States	9 Cycles	4 Cycles
19 μs	22.5 μs	4 μs

b.) Ouput Routine

8085		8048		8051	
LXI	HL,SERBUF	MOV	RO,#SERBUF		
MOV	A,M	MOV	A,@RO	MOV	A,SERBUF
RR		RRC	A	RRC	A
MOV	M,A	MOV	@RO,A	MOV	SERBUF,A
IN	SERPORT				
JC	HI	JC	HI		
LO: ANI	NOT MASK	ANL	SERPRT,#NOT MASK	MOV	SERPIN,C
JMP	CNT	JMP	CNT		
HI: ORI	MASK	HI: ORL	SERPRT,#MASK		
CNT: OUT	SERPORT	CNT:			

RESULTS:

10 Instructions	8 Instructions	4 Instructions
20 Bytes	13 Bytes	7 Bytes
72 States	11 Cycles	5 Cycles
24 μs	27.5 μs	5 μs

Design Example #3 — Combinatorial Logic Equations

Some simple uses for bit-test instructions and logical operations follow.

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 6-12 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation

$$Q = (U \cdot (V + W)) + (X \cdot \overline{Y}) + \overline{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.

For the sake of comparison, this function is implemented three ways, restricting the software to three proper subsets of the 8051 Family instruction set. It is also assumed that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 6-13. Program flow would embark on a routine down a test-and-branch tree and leaves either the "True" or "Not True" exit as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, no generalized bit-addressing instructions are used. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. 8051 Family mnemonics are used in Example 6-2a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.
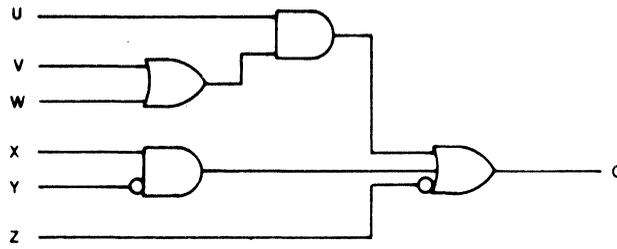
$$Q = (U \bullet (V + W)) + (X \bullet \overline{Y}) + \overline{Z}$$

a. Using TTL
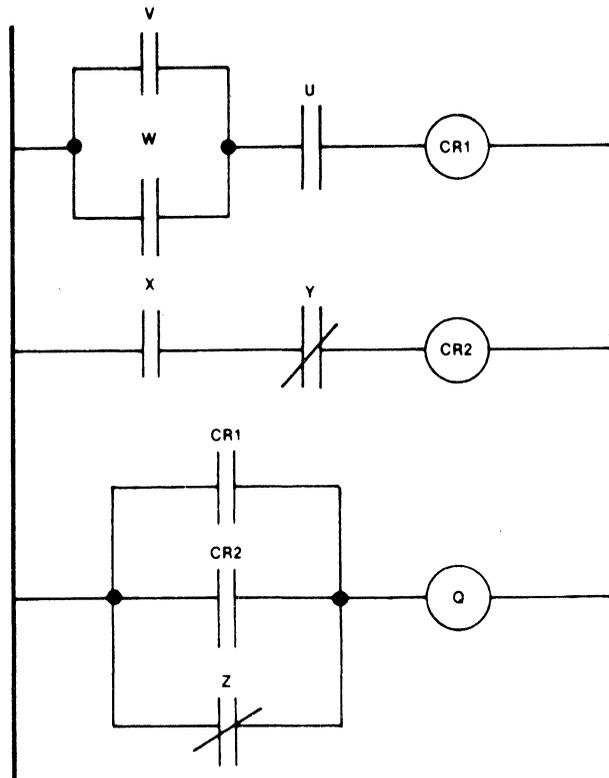
b. Using Relay Logic

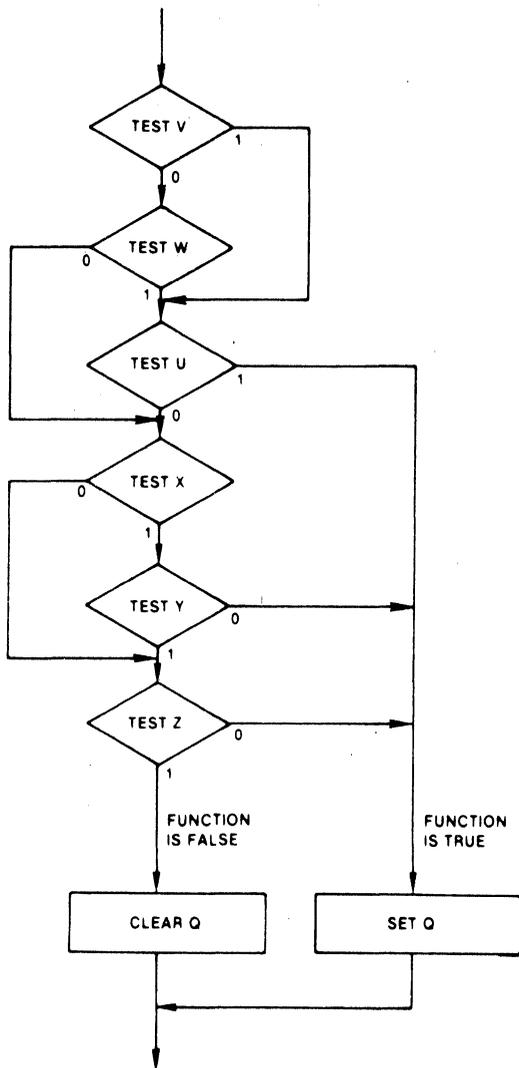Figure 6-12. Hardware Implementations of Boolean Functions

Figure 6-13. Flow Chart for Tree-Branching Algorithm

The code that results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time varies widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move mask conditional jump sequence in Example 6-2a, but the algorithm would be equally convoluted (see Example 6-2b). To lessen the confusion, "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 6-2c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code. When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed — code can be written directly from the logic diagrams in Figure 6-12. The result is simplicity itself; fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N + 1 lines of source program — one for each input and one line to store the results. To simulate NAND or NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles," perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 6-2. Software Solutions to Logic Function of Figure 6-12.

a. Using only byte-wide logical instructions.

```
; BUFNCI  SOLVE RANDOM LOGIC FUNCTION
;         OF 6 VARIABLES BY LOADING AND
;         MASKING THE APPROPRIATE BITS
;         IN THE ACCUMULATOR, THEN
;         EXECUTING CONDITIONAL JUMPS
;         BASED ON ZERO CONDITION.
;         (APPROACH USED BY BYTE-
;         ORIENTED ARCHITECTURES.)
;         BYTE AND MASK VALUES
;         CORRESPOND TO RESPECTIVE BYTE
;         ADDRESS AND BIT POSITIONS.
;
OUTBUF   DATA 22H  ;OUTPUT PIN STATE MAP
;
TESTV:   MOV      A,P2
         ANL      A,#00000100B
         JNZ      TESTU
         MOV      A,TCON
         ANL      A,#00100000B
         JZ       TESTX
TESTU:   MOV      A,P1
         ANL      A,#00000010B
         JNZ      SETQ
```

```
TESTX:    MOV    A,TCON
          ANL    A,#00001000B
          JZ     TESTZ
          MOV    A,20H
          ANL    A,#00000001B
          JZ     SETQ
TESTZ:    MOV    A,21H
          ANL    A,#00000010B
          JZ     SETQ
CLRQ:     MOV    A,OUTBUF
          ANL    A,#11110111B
          JMP    OUTQ
SETQ:     MOV    A,OUTBUF
          ORL    A,#00001000B
OUTQ:     MOV    OUTBUF,A
          MOV    P3,A
```

b. Using only bit-test Instructions.

```
; BFUNC2    SOLVE RANDOM LOGIC FUNCTION
;           OF 6 VARIABLES BY DIRECTLY
;           POLLING EACH BIT.
;           (APPROACH USING 8051-FAMILY UNIQUE
;           BIT-TEST INSTRUCTION CAPABILITY.)
;           SYMBOLS USED IN LOGIC DIAGRAM
;           ASSIGNED TO CORRESPONDING 8x51
;           BIT ADDRESSES.
;
U          BIT    P1.1
V          BIT    P2.2
W          BIT    TF0
X          BIT    1E1
Y          BIT    20H.0
Z          BIT    21H.1
Q          BIT    P3.3
;          ...    .....
TEST_V:   JB     V,TEST_U
          JNB    W,TEST_X
TEST_U:   JB     U,SET_Q
TEST_X:   JNB    X,TEST_Z
          JNB    Y,SET_Q
TEST_Z:   JNB    Z,SET_Q
```

```
CLR_Q:    CLR    Q
          JMP    NXTTST
SET_Q:    SETB   Q
NXTTST:          ;CONTINUATION OF PROGRAM
```

c. Using logical operations on Boolean variables.

```
; FUNC3    SOLVE A RANDOM LOGIC FUNCTION
;          OF 6 VARIABLES USING
;          STRAIGHT_LINE LOGICAL
;          INSTRUCTIONS ON 8051 BOOLEAN
;          VARIABLES.
;

MOV    C,V
ORL    C,W      ; OUTPUT OF OR GATE
ANL    C,U      ; OUTPUT OF TOP AND GATE
MOV    F0,C     ; SAVE INTERMEDIATE STATE
MOV    C,X
ANL    C,Y      ; OUTPUT OF BOTTOM AND GATE
ORL    C,F0     ; INCLUDE VALUE SAVED ABOVE
ORL    C,Z      ; INCLUDE LAST INPUT VARIABLE
MOV    Q,C      ; OUTPUT COMPUTED RESULT
```

An upper limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown above, The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

Design Example #4 — Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the 3-position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down position, contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly — unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6-5 summarizes these operating modes.

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 6-14 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated, the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions.

A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, it shows that the hardest part of many system designs is determining what the controller should do. Writing the software to solve these functions is comparatively easy. Secondly, it shows the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical parts count.

Table 6-5. Truth Table for Turn-Signal Operation

| Input Signals				Output Signals			
Brake Switch	Emerg. Switch	Left Turn Switch	Right Turn Switch	Left Front & Dash	Right Front & Dash	Left Rear	Right Rear
0	0	0	0	Off	Off	Off	Off
0	0	0	1	Off	Blink	Off	Blink
0	0	1	0	Blink	Off	Blink	Off
0	1	0	0	Blink	Blink	Blink	Blink
0	1	0	1	Blink	Blink	Blink	Blink
0	1	1	0	Blink	Blink	Blink	Blink
1	0	0	0	Off	Off	On	On
1	0	0	1	Off	Blink	On	Blink
1	0	1	0	Blink	Off	Blink	On
1	1	0	0	Blink	Blink	On	On
1	1	0	1	Blink	Blink	On	Blink
1	1	1	0	Blink	Blink	Blink	On
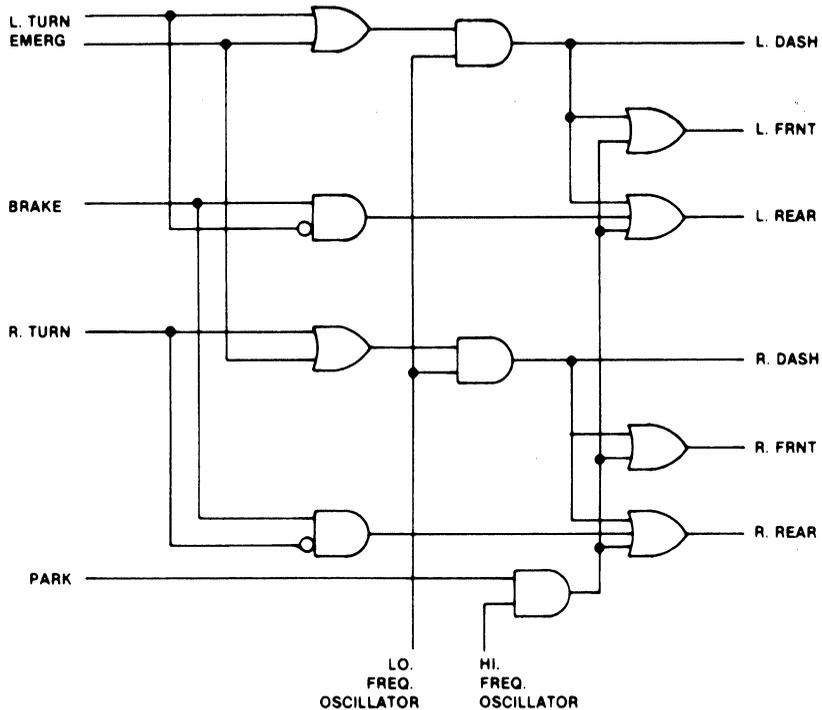
Figure 6-14. TTL Logic Implementation of Automotive Turn Signals

The Single-Chip Solution

The circuit shown in Figure 6-15 indicates five input pins to the five input variables — left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```
;
;           INPUT PIN DECLARATIONS:
;           (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
;
BRAKE       BIT   P1.0    ; BRAKE PEDAL DEPRESSED
```

```
EMERG       BIT   P1.1    ; EMERGENCY BLINKER
                            ACTIVATED

PARK        BIT   P1.2    ; PARKING LIGHTS ON

L_TURN      BIT   P1.3    ; TURN LEVER DOWN

R_TURN      BIT   P1.4    ; TURN LEVER UP

;

;           OUTPUT PIN DECLARATIONS

;

L_FRNT      BIT   P1.5    ; FRONT LEFT-TURN
                            INDICATOR

R_FRNT      BIT   P1.6    ; FRONT RIGHT-TURN
                            INDICATOR

L_DASH      BIT   P1.7    ; DASHBOARD LEFT-TURN
                            INDICATOR

R_DASH      BIT   P2.0    ; DASHBOARD RIGHT-TURN
                            INDICATOR

L_REAR      BIT   P2.1    ; REAR LEFT-TURN
                            INDICATOR

R_REAR      BIT   P2.2    ; REAR RIGHT-TURN
                            INDICATOR
```
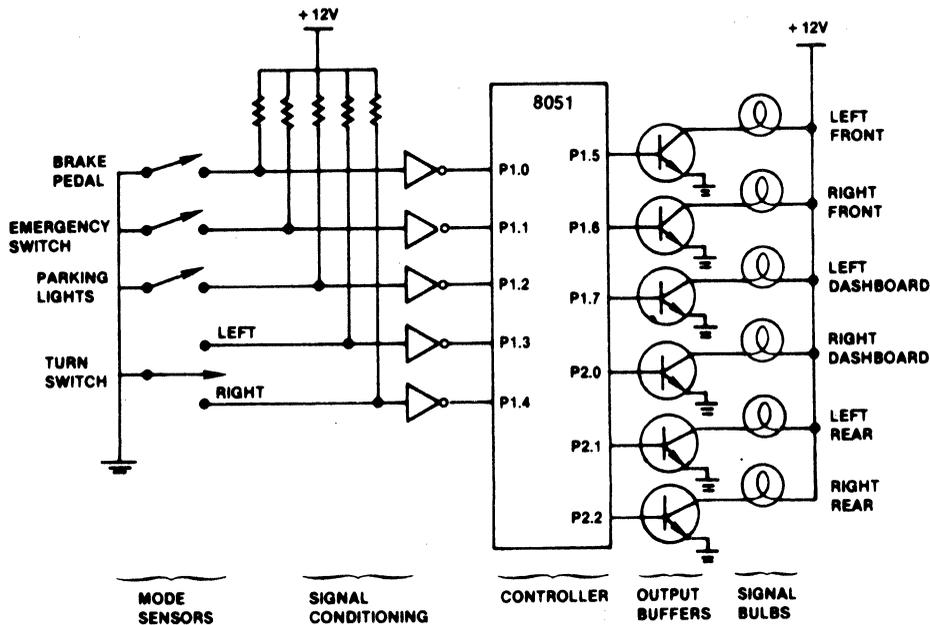
Figure 6-15. Microcomputer Turn-Signal Connections

Another key advantage of symbolic addressing appears further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, and heat sinks, etc., all affect PC board layout. It is quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures use mask bytes or "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from P1.1, for example, to P3.4) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes trivial. The number of the port containing the pin is irrelevant, and masks and complex program structures are not needed. Only the initial Boolean variable declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

```
;              ...           .....
; INTERRUPT RATE SUBDIVIDER
SUB_DIV    DATA       20H
; HIGH-FREQUENCY OSCILLATOR BIT
HI_FREQ    BIT        SUB_DIV.0
; LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ    BIT        SUB-DIV.7
;          ...
           ORG        0000H
JMP        INIT
           ...        .....
           ORG        100H
; PUT TIMER 0 IN MODE 1
INIT:      MOV        TMOD,#00000001B
; INITIALIZE TIMER REGISTERS
           MOV        TL0,#0
           MOV        TH0,#-16
; SUBDIVIDE INTERRUPT RATE BY 244
           MOV        SUB_DIV,#244
; ENABLE TIMER INTERRUPTS
           SETB       ET0
```

```
; GLOBALLY ENABLE ALL INTERRUPTS
          SETB      EA
; START TIMER
          SETB      TR0
;
; (CONTINUE WITH BACKGROUND PROGRAM)
;
; PUT TIMER 0 IN MODE 1
; INITIALIZE TIMER REGISTERS

; SUBDIVIDE INTERRUPT RATE BY 244
; ENABLE TIMER INTERRUPTS
; GLOBALLY ENABLE ALL INTERRUPTS
; START TIMER
```

Timer 0 (one of the two on-chip timer/counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization, it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 µs. Timer-interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows.

An 8-bit variable in the bit-addressable RAM array is needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very fast to modulate the parking lights; bit 7 is "turned" to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4,096 ms. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV with 244 initially and each time it decrements to zero, will produce a 0.999 second period for the highest-order bit.

```
ORG    000BH       ; TIMER 0 SERVICE VECTOR
MOV    TH0,#-16
PUSH   PSW
PUSH   ACC
PUSH   B
DJNZ   SUB_DIV,T0SERV
MOV    SUB_DIV,#244
```

The code to sample inputs, performs calculations, and update outputs — the real essence of the signal-controller algorithm — may be performed either as part of the

interrupt-service routine or as part of a background-program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 6-14) that the subterm (PARK · H_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named "DIM". The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since the PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```
DIM    BIT     PSW.1   ; DECLARE TEMP STORAGE
                               FLAG
; ...   .....
MOV    C,PARK          ; GATE PARKING LIGHT
                               SWITCH
ANL    HI_FREQ         ; WITH HIGH FREQUENCY
                               SIGNAL
MOV    DIM,C           ; AND SAVE IN TEMP
                               VARIABLE.
```

This simple 3-line selection of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, independent of the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```
;
MOV    C,L_TURN     ; SET CARRY IF TURN
ORL    C,EMERG      ; OR EMERGENCY SELECTED.
ANL    C,LO_FREQ    ; GATE IN 1 HZ SIGNAL
MOV    L_DASH,C     ; AND OUTPUT TO DASHBOARD.
```

To generate the left-front turn signal, we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in F0.

```
;
MOV    F0,C        ; SAVE FUNCTION SO FAR.
ORL    C,DIM       ; ADD IN PARKING LIGHT FUNCTION
MOV    L_FRNT,C    ; AND OUTPUT TO TURN SIGNAL.
```

Finally, the rear left-turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```
MOV   C,BRAKE     ; GATE BRAKE PEDAL SWITCH
ANL   C,L_TURN    ; WITH TURN LEVER.
ORL   C,F0        ; INCLUDE TEMP. VARIABLE
                    FROM DASH
ORL   C,DIM       ; AND PARKING LIGHT FUNCTION
MOV   L_REAR,C    ; AND OUTPUT TO TURN SIGNAL
```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```
MOV   C,R_TURN    ; SET CARRY  IF TURN
ORL   C,EMERG     ; OR EMERGENCY SELECTED.
ANL   C,LO_FREQ   ; IF SO, GATE IN 1 HZ SIGNAL
MOV   R_DASH,C    ; AND OUTPUT TO DASHBOARD.
MOV   F0,C        ; SAVE FUNCTION SO FAR.
ORL   C,DIM       ; ADD IN PARKING LIGHT
                    FUNCTION
MOV   R_FRNT,C    ; AND OUTPUT TO TURN SIGNAL.
MOV   C,BRAKE     ; GATE BRAKE PEDAL SWITCH
ANL   C,R_TURN    ; WITH TURN LEVER.
ORL   C,F0        ; INCLUDE TEMP.VARIABLE FROM
                    DASH
ORL   C,DIM       ; AND PARKING LIGHT FUNCTION
MOV   R_REAR,C    ; AND OUTPUT TO TURN SIGNAL.
```

The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

```
POP   B           ; RESTORE CPU REGISTERS.
POP   ACC
POP   PSW
RETI
```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear; the 50% duty cycle of HI_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc., are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 Hz signals of seven different duty cycles could be produced by considering bits 2-0 as shown in Table 6-6. The only software change required would be to the code which sets-up variable DIM:

```
MOV   C,SUB_DIV.1   ; START WITH 50 PERCENT
ANL   C,SUB_DIV.0   ; MASK DOWN TO 25
ORL   C,SUB_DIV.2   ; AND BUILD BACK TO 62
                      PERCENT
MOV   DIM,C         ; DUTY CYCLE FOR PARKING
                      LIGHTS.
```

Table 6-6. Non-trivial Duty Cycles

| Sub_Div Bits | | | | | | | | Duty Cycles | | | | | | |
7	6	5	4	3	2	1	0	12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0	Off	Off	Off	Off	Off	Off	Off
X	X	X	X	X	0	0	1	Off	Off	Off	Off	Off	Off	On
X	X	X	X	X	0	1	0	Off	Off	Off	Off	Off	On	On
X	X	X	X	X	0	1	1	Off	Off	Off	Off	On	On	On
X	X	X	X	X	1	0	0	Off	Off	Off	On	On	On	On
X	X	X	X	X	1	0	1	Off	Off	On	On	On	On	On
X	X	X	X	X	1	1	0	Off	On	On	On	On	On	On
X	X	X	X	X	1	1	1	On	On	On	On	On	On	On

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 6-15 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 6-16) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs are available.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is 1 Mb/s, or 8 μs per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated, the B register is shifted by the serial port to the shift-register drive. The outputs may glitch as data is shifted through them; at 1 Mb/s, however, the results (blinking lights) will not be noticed. Many shift registers provide an "enable" bit to hold the output states while new data is being shifted in.

This is where the earlier decision to address bits symbolically throughout the program pays off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

L_FRNT	BIT	B.0	; FRONT LEFT-TURN INDICATOR
R_FRNT	BIT	B.1	; FRONT RIGHT-TURN INDICATOR
L_DASH	BIT	B.2	; DASHBOARD LEFT-TURN INDICATOR
R_DASH	BIT	B.3	; DASHBOARD RIGHT-TURN INDICATOR
L_REAR	BIT	B.4	; REAR LEFT-TURN INDICATOR
R_REAR	BIT	B.5	; REAR RIGHT-TURN INDICATOR

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port:

MOV   SBUF,B   ; LOAD BUFFER AND TRANSMIT

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than ten. The wiring harness is, therefore, simpler and less expensive — one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and or fault detection on the four main turn indicators. Each could *still* be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.
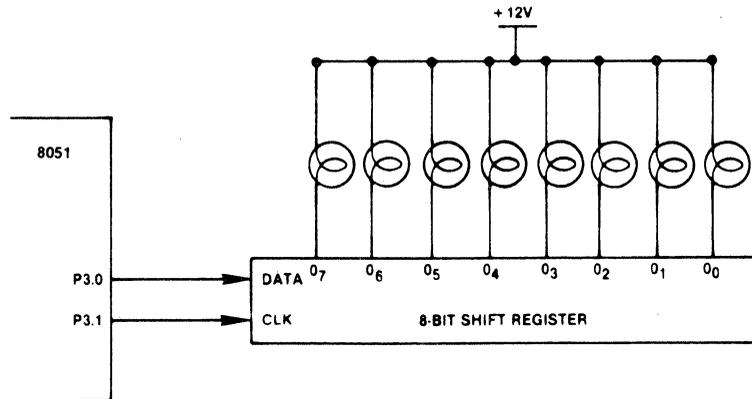
Figure 6-16. Output Expansion Using Serial Port

Even with redundancy, the lights will eventually fail. To handle this inescapable fact, current or voltage sensing circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 6-17 shows one such circuit.

Assume all of the lights are turned on except one, i.e., all but one of the collectors are grounded. For the bulb that is turned off, if there is continuity from + 12 V through the bulb base and filament, the control wire, all connectors, and the PC boards traces; and if the transistor is indeed not shorted to ground, then the collector will be pulled to + 12 V. This turns on the base of Q7 through the

corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin (T0) should be high. However, a control wire shorted to + 12 V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine.
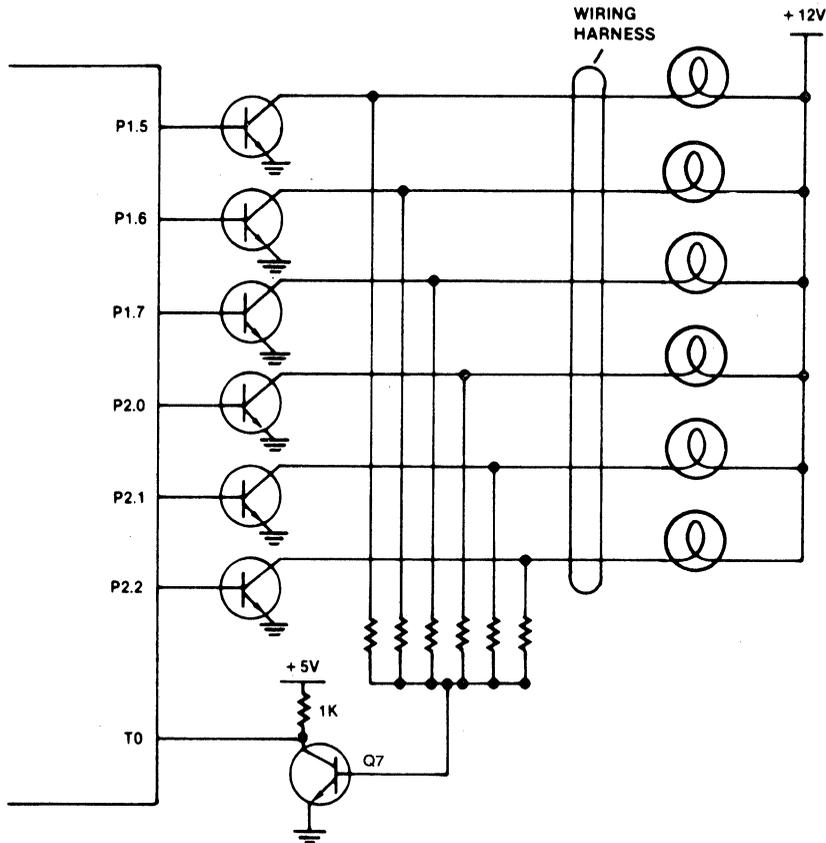
Figure 6-17. Fault Detection

```
        DJNZ    SUB_DIV,T0SERV
        MOV     SUB_DIV,#244    ; RELOAD COUNTER
        ORL     P2,#11100000B   ; SET CONTROL OUTPUTS
                                  HIGH
        ORL     P2,#00000111B
        CLR     L_FRNT          ; FLOAT DRIVE COLLECTOR
        JB      T0,FAULT        ; T0 SHOULD BE PULLED
                                  LOW
        SETB    L_FRNT          ; PULL COLLECTOR BACK
                                  DOWN
        CLR     L_DASH
        JB      T0,FAULT
        SETB    L_DASH
        CLR     L_REAR
        JB      T0,FAULT
        SETB    L_REAR
        CLR     R_FRNT
        JB      T0,FAULT
        SETB    R_FRNT
        CLR     R_DASH
        JB      T0,FAULT
        SETB    R_DASH
        CLR     R_REAR
        JB      T0,FAULT
        SETB    R_REAR
;
; WITH ALL COLLECTORS GROUNDED, T0 SHOULD BE HIGH
; IF SO, CONTINUE WITH INTERRUPT ROUTINE.
        JB      T0,T0SERV
FAULT:                          ; ELECTRICAL FAILURE
                                ; PROCESSING ROUTINE
T0SERV:                         ; CONTINUE WITH
                                  INTERRUPT PROCESSING
```

The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidentally) 67 µs, plus 32 µs once per second for the electrical test. If executed every 4 ms as suggested, this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation.

Design Example #5 — Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of the 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 6-18 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors;
- 12 output signals;
- Combinational and sequential logic computations;
- Remote operation with communications to a host processor via a high-speed full-duplex serial link;
- Two prioritized external interrupts;
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs *no* other integrated circuits!

The 64 input sensors are logically arranged as an 8 x 8 matrix, The pins of Port 1 sequentially enable each column of the sensor matrix; as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.
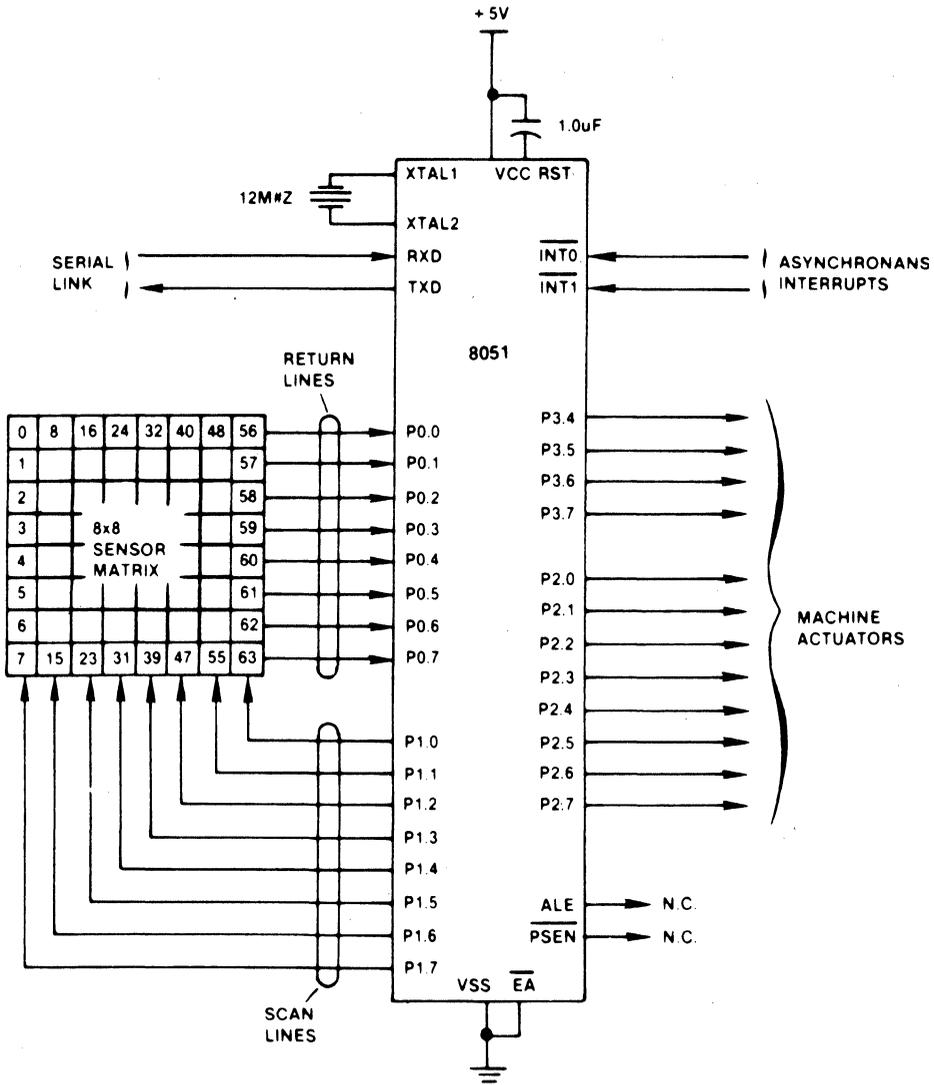
Figure 6-18. Block Diagram of 64-Input Machine Controller

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 6–19a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kΩ resistors and are read as zeros. The resistor values must be chosen to ensure all return lines are pulled above the 2.0 V logic threshold, even in the worst case, where all contacts in an enabled column are closed. Since P0 is provided open-collector outputs and high-impedance MOS inputs, its input loading may be considered negligible.

The circuits in Figures 6-19b and d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs *and* provide optical isolation as in Figure 6–19b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 6-19c uses octal 3-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. The 8-to-1 multiplexers in Figure 6-19d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. Five more output pins are thus freed for more control functions. Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 6-20 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable

region of the RAM; one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and or debounce contact closures by comparing each bit with its earlier value.

The code in Example 6-3 implements the scanning algorithm for the circuits in Figure 6-19. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic; ones represent contacts that are closed or isolators turned on.

Example 6-3.

INPUT—SCAN:			; SUBROUTINE TO READ CURRENT STATE OF 64 SENSORS AND SAVE IN RAM 20H-27H.
	MOV	R0,#20H	; INITIALIZE POINTERS
	MOV	R1,#28H	; FOR BIT MAP BASES.
	MOV	A,#80H	; SET FIRST BIT IN ACC.
SCAN:	MOV	P1,A	; OUTPUT TO SCAN LINES.
	RR	A	; SHIFT TO ENABLE NEXT COLUMN NEXT.
	MOV	R2,A	; REMEMBER CURRENT SCAN POSITION.
	MOV	A,P0	; READ RETURN LINES.
	XCH	A,@R0	; SWITCH WITH PREVIOUS MAP BITS.
	MOV	@R1,A	; SAVE PREVIOUS STATE AS WELL.
	INC	R0	; BUMP POINTERS.
	INC	R1	
	MOV	A,R2	; RELOAD SCAN LINE MASK
	JNB	ACC.7,SCAN	; LOOP UNTIL ALL EIGHT COLUMNS READ.
	RET		

What happens after the sensors have been scanned depends on the individual application. Rather than inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.
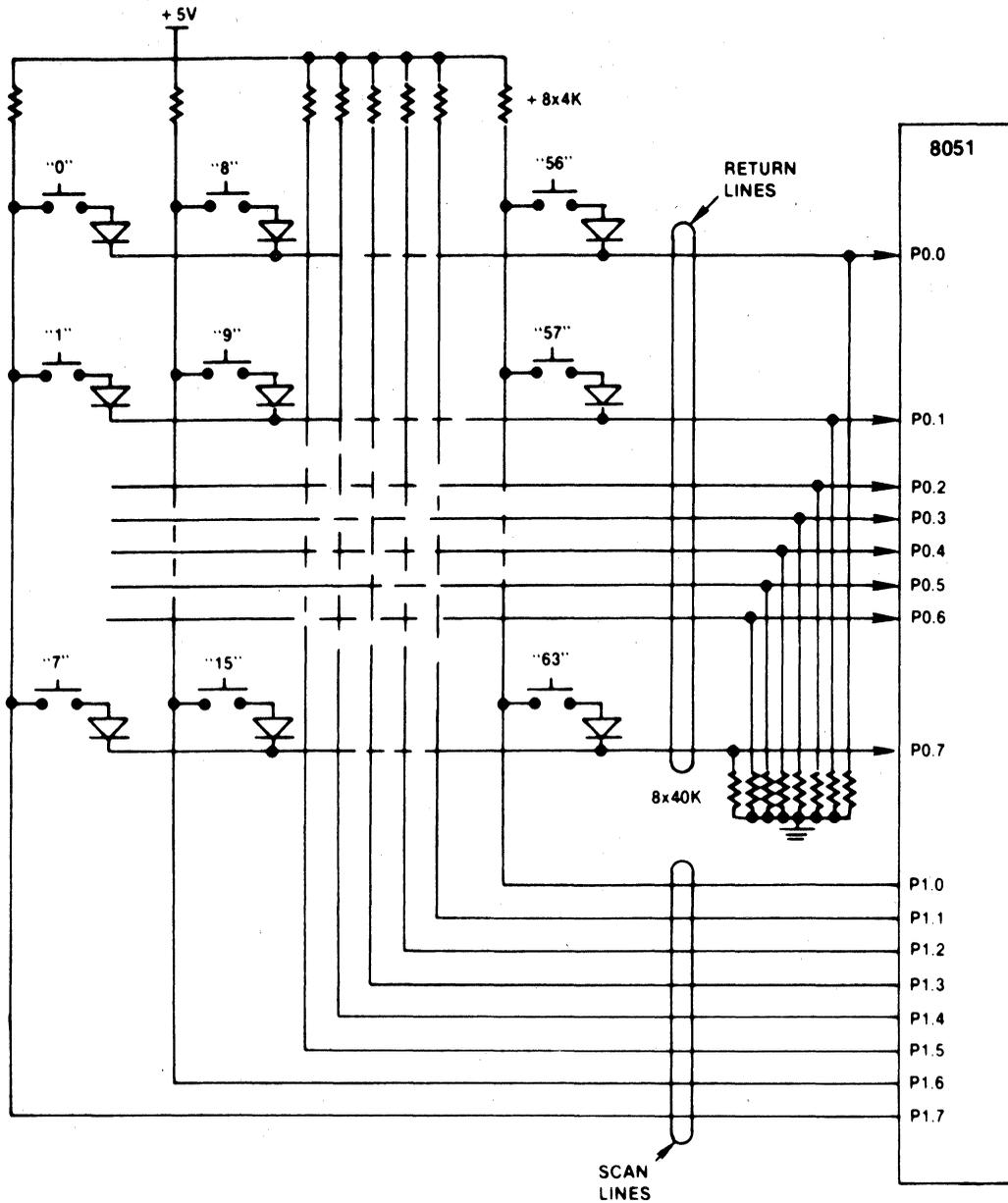
a. Using Switch Contact/Diode Matrix

Figure 6-19. Sensor Matrix Implementation Methods
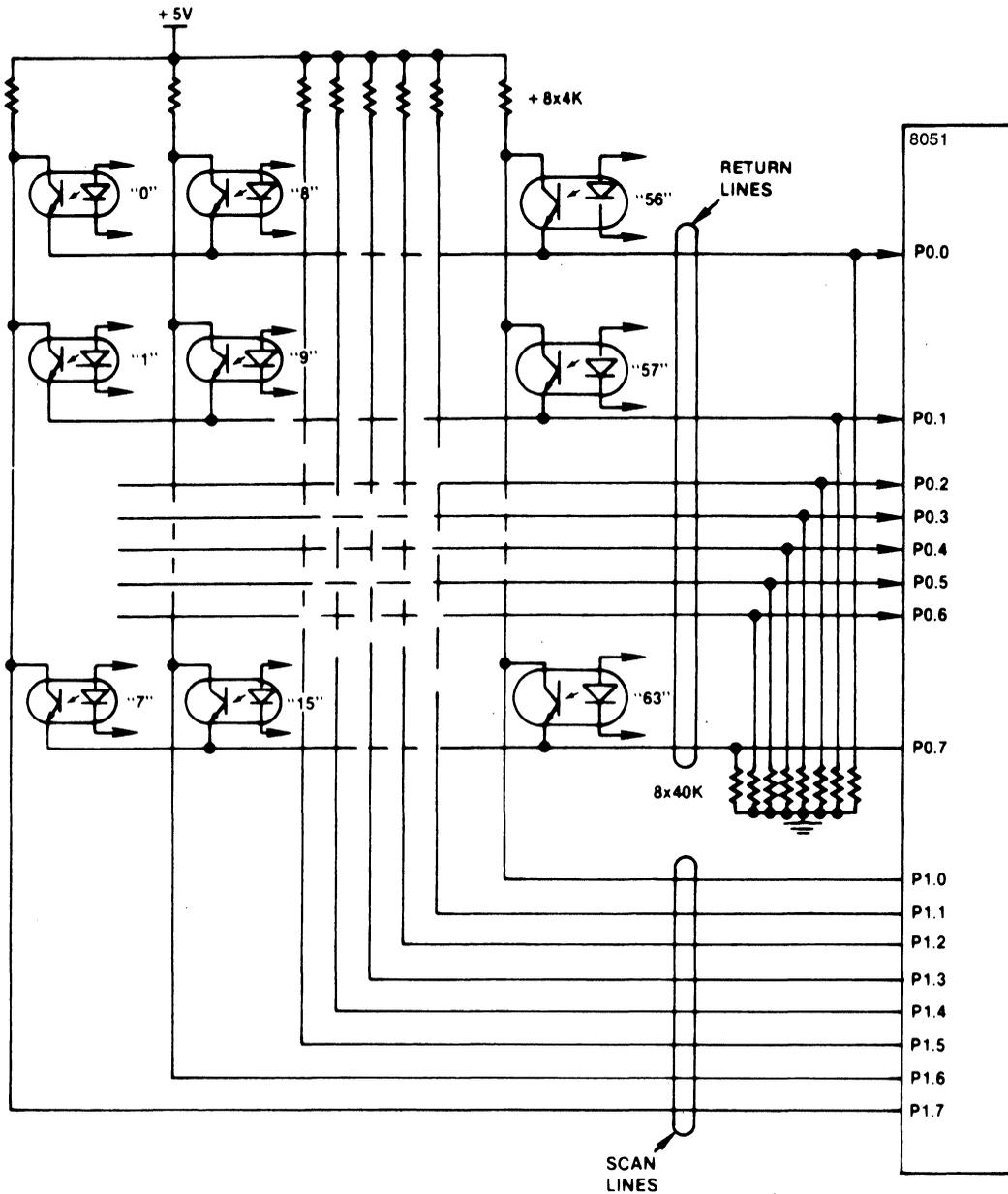
b. Using Optically-Coupled Isolators

Figure 6-19. Sensor Matrix Implementation Methods (continued)
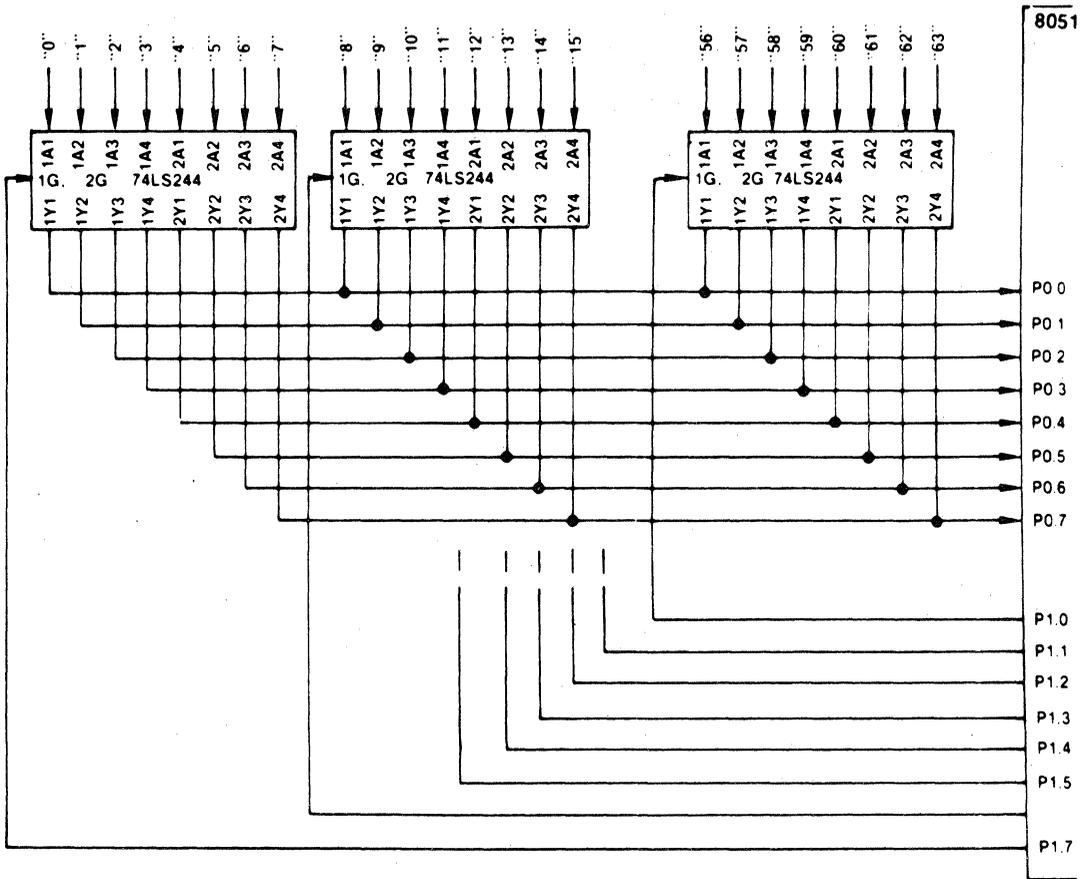
c. Using TTL Three-State Buffers

Figure 6-19. Sensor Matrix Implementation Methods (continued)
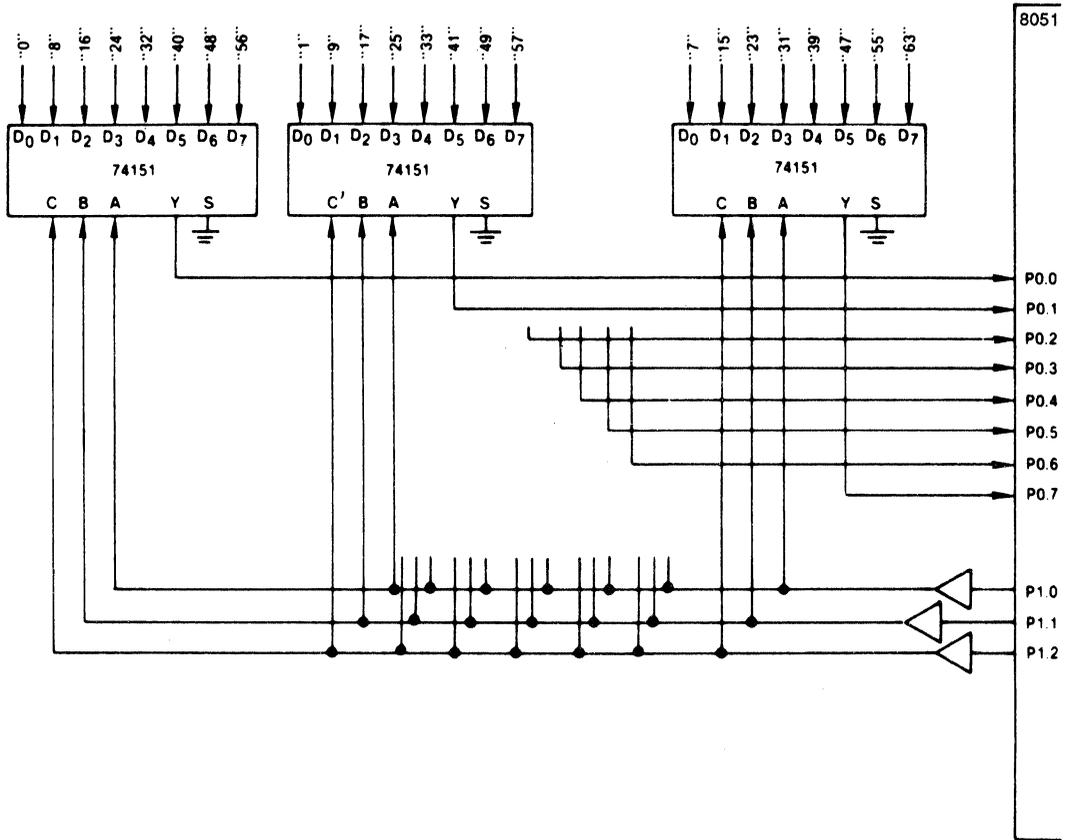
d. Using TTL Data Selectors

Figure 6-19. Sensor Matrix Implementation Methods (continued)
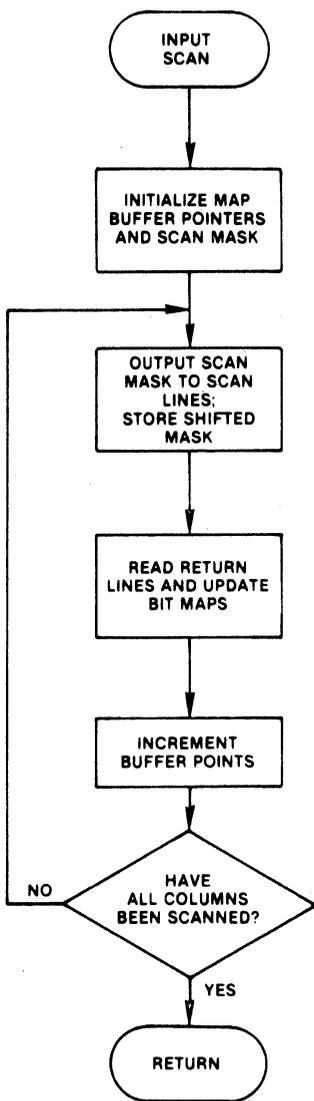
Figure 6-20. Flowchart for Reading in Sensor Matrix

Combinatorial Output Variables. An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example #3. All 64 inputs are represented in the bit maps; in fact, the sensor numbers in Figure 6-19 correspond to the absolute bit addresses in RAM! The code in Example 6-4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 6-4. Simple Combinatorial Output Variables.

```
; SET P2.2 = (12) (23) (34) (45) (56)
    MOV    C,12
    ANL    C,23
    ANL    C,34
    ANL    C, 45
    ANL    C, 56
    MOV    P2.2,C
```

Intermediate Variables. The examination of a typical relay-logic ladder diagram will show that many of the rungs control are *not* outputs, but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The 8051 Family solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 6-21 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and saved (as implied by the diagram). As the program proceeds this bit can qualify each output.

Figure 6-21. Ladder Diagram for Output Override Circuitry

Example 6-5. Incorporating Override signal into actuator outputs.

```
        CALL    INPUT_SCAN
        MOV     C,0
        ORL     C,1
        ORL     C,2
        ORL     C,3
        MOV     F0,C
        ....    .....
;       COMPUTE FUNCTION 0
;
        ANL     C,F0
        MOV     P1.0,C
;       ....    .....
;       COMPUTE FUNCTION 1
;
        ANL     C,F0
        MOV     P1.1,C
;       ....    .....
```

```
;       COMPUTE FUNCTION 2
;
        ANL     C,F0
        MOV     P1.2,C
;       ....    .....
```

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state — analogous to a TTL Set-Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an "emergency cleared" button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6-6).

Example 6-6. Simulating a latching relay.

```
; L_SET    SET FLAG 0 IF C = 1
L_SET:      ORL     C,F0
            MOV     F0,C
;
; L_RESET   RESET FLAG 0 IF C = 1
L_RESET     CPS     C
            ANL     C,F0
            MOV     F0,C
```

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a dc motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer/counters. The procedure followed by the routine depends heavily on the details of the exact function needed; time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 ms the code in Example 6-7 will clear an intermediate variable set by the background program after it has been active for 2 s.

Example 6-7. Code to clear USRFLG after a fixed time delay.

```
            JNB     USR_FLG,NXTTST
            DJNZ    DLAY_COUNT,NXTTST
            CLR     USR_FLG
            MOV     DLAY_COUNT,#200
NXTTST:     ....    .....
```

Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate.

Response Timing. One difference between relay and programmed industrial controllers (when each is considered as a "black box") is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this chapter completes in 1 or 2 µs at 12 MHz* — the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computation would require about a millisecond; less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, or perform system diagnostics.

Additional functions and uses

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

```
; EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY
; USING F0 AS INPUT VARIABLE.
XOR_F0:    JNB    F0,XORCNT    ; ("JB" FOR X-NOR)
           CPL    C
XORCNT:    ...    ......
```

XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the rotate-through-carry instructions, though this would alter the accumulator data.

```
; EXCHANGE CARRY WITH USRFLG
XCHBIT:    RLC    A
           MOV    C,USR_FLG
           RRC    A
           MOV    USR_FLG,C
           RLC    A
```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 6-2b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 µs — still much faster than most architectures — or used for logical calculations. This technique can also simulate additional bit addressing modes with byte operations.

Parity of bytes or bits. The parity of the current accumulators contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes. Though the 8051 serial port can accommodate 8- or 9-bit data transmissions, some protocols involve much longer bit streams. The algorithms presented in Design Example 6-2 can be extended quite readily to 16 or more bits by using multibyte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte date buffers are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

SUMMARY

A unique facet of the 8051 Family microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct-addressing modes, program and mass-data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This chapter has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design; more reliable to implement, debug, and verify; use less program memory; and run up to an order-of-magnitude faster than the same function implemented on previous digital-computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect; the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some ways from this duality. Data-intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

SECTION II

8051 Family Device Description

Section II contains the data sheets, device-specific application information, software routines, third-party development support, and package outlines.

The data sheets are divided into three chapters corresponding to three product families. In general, devices are listed in order of increasing functionality. EPROM data sheets follow the ROM data sheets with which they are associated.

Application information and software routines immediately follow the data sheets for which they are most closely intended, although they will also be of use with data sheets of more enhanced devices.
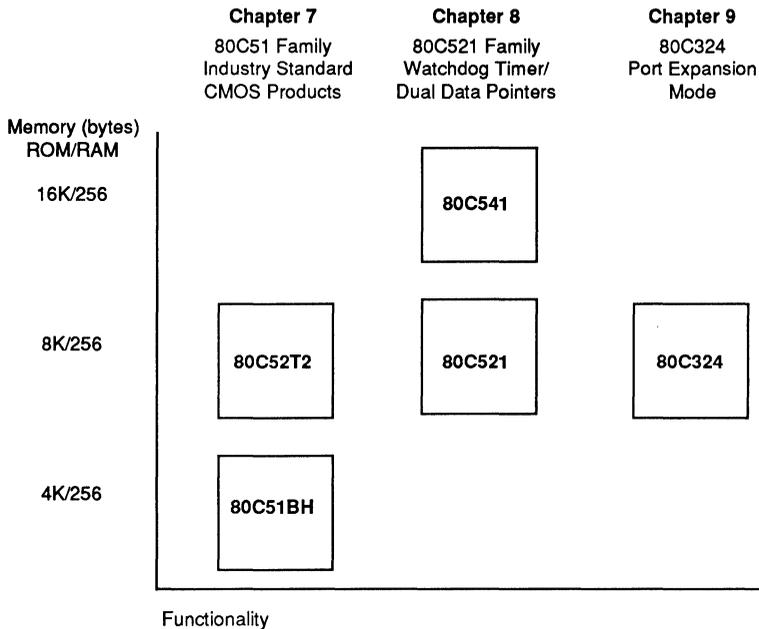
	Chapter 7	Chapter 8	Chapter 9
	80C51 Family Industry Standard CMOS Products	80C521 Family Watchdog Timer/ Dual Data Pointers	80C324 Port Expansion Mode

Memory (bytes)
ROM/RAM

16K/256		80C541	
8K/256	80C52T2	80C521	80C324
4K/256	80C51BH		

Functionality

CHAPTER 7

80C51BH/80C31BH/80C52T2/80C32T2

CMOS Single-Chip Microcontrollers

DISTINCTIVE CHARACTERISTICS

- Industry Standard CMOS Microcontrollers
- Low Power Modes—Idle & Power-Down
- 32 Programmable I/O Lines
- Two 16-bit Counter/Timers
- Programmable Serial Channel
  - Five-source, two-level Interrupt Structure
  - Boolean Processor
- 64K bytes Program Memory Space
- 64K bytes Data Memory Space

	RAM (bytes)	ROM (bytes)
80C31BH	128	—
80C51BH	128	4K
80C32T2	256	—
80C52T2	256	8K

80C51BH = 80C31BH + 4K bytes ROM
80C52T2 = 80C32T2 + 8K bytes ROM

GENERAL DESCRIPTION

The 80C51BH and 80C31BH are CMOS versions of the industry-standard 8051 architecture. The 80C52T2 and 80C32T2 are identical products except they contain double the on-chip memory.

Both the 80C51BH and 80C31BH include 128 bytes of RAM, while the 80C52T2 and 80C32T2 include 256 bytes of RAM. The 80C51BH also includes 4K bytes of custom ROM program memory and the 80C52T2 includes 8K bytes of ROM. The 80C52T2 and 80C32T2 are CMOS equivalents to the 8052AH and 8032AH except they contain two timers (T2) instead of three.

These CMOS products retain all of the features of their NMOS counterparts: 32 I/O lines; two 16-bit counter/

timers; a full-duplex serial port; a five-source, two-level interrupt structure; and an on-chip oscillator and clock circuits.

In addition, all CMOS 80C51-based products have two software-selectable modes of reduced activity for further power conservation—Idle and Power-Down. In the Idle mode, the CPU is frozen while the RAM, timers, serial port, and interrupt system continue to function. In the Power-Down mode, the RAM is saved and all other functions are inoperative.

The 80C52T2 and 80C32T2 in PLCC packages offer improved noise tolerance by utilizing previously unused pins for additional V_{CC} and V_{SS} connections.

BLOCK DIAGRAM
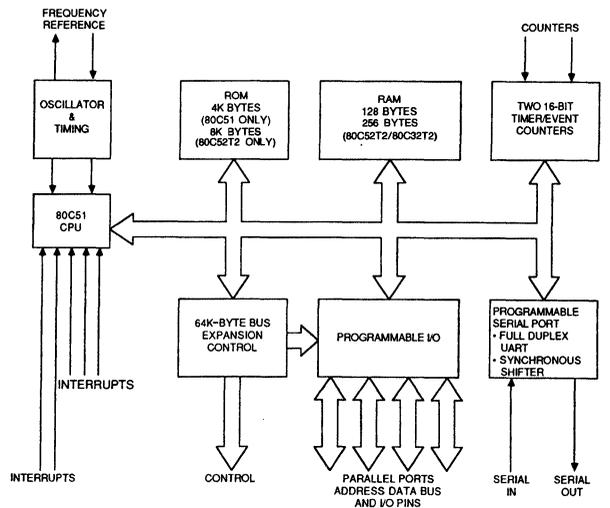
BD007232

CONNECTION DIAGRAMS
Top View

DIP
80C51BH/80C31BH
80C52T2/80C32T2

P1.0	1	40	V_CC
P1.1	2	39	P0.0 AD_0
P1.2	3	38	P0.1 AD_1
P1.3	4	37	P0.2 AD_2
P1.4	5	36	P0.3 AD_3
P1.5	6	35	P0.4 AD_4
P1.6	7	34	P0.5 AD_5
P1.7	8	33	P0.6 AD_6
RST	9	32	P0.7 AD_7
RXD P3.0	10	31	$\overline{EA}$
TXD P3.1	11	30	ALE
$\overline{INT_0}$ P3.2	12	29	$\overline{PSEN}$
$\overline{INT_1}$ P3.3	13	28	P2.7 A_15
T_0 P3.4	14	27	P2.6 A_14
T_1 P3.5	15	26	P2.5 A_13
$\overline{WR}$ P3.6	16	25	P2.4 A_12
$\overline{RD}$ P3.7	17	24	P2.3 A_11
XTAL_2	18	23	P2.2 A_10
XTAL_1	19	22	P2.1 A_9
V_SS	20	21	P2.0 A_8
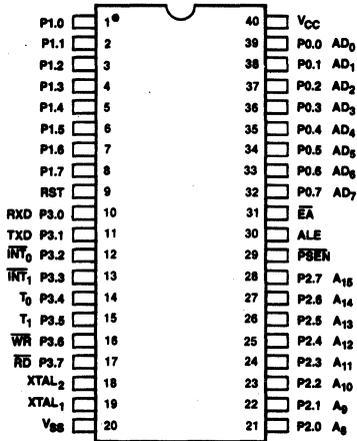
CD005554

PLCC
80C51BH/80C31BH
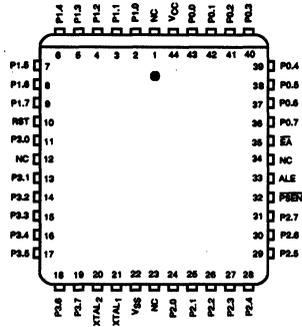
CD009443

PLCC
80C52T2/80C32T2

CD009444

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL

LS001323

ORDERING INFORMATION

Commodity Products

AMD commodity products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:
  a. **Temperature Range**
  b. **Package Type**
  c. **Device Number**
  d. **Speed Option**
  e. **Optional Processing**

```
  P              80C51BH

                                         e. OPTIONAL PROCESSING
                                            Blank = Standard processing

                                    d. SPEED OPTION
                                       Blank = 0.1 to 12 MHz
                                         -1 = 0.1 to 16 MHz
                                        -20 = 0.1 to 20 MHz (preliminary)

                          c. DEVICE NUMBER/DESCRIPTION
                             80C51BH/80C31BH/80C52T2/80C32T2
                             CMOS Single-Chip Microcontrollers

            b. PACKAGE TYPE
               P = 40-Pin Plastic DIP (PD 040)
               N = 44-Pin Plastic Leaded Chip Carrier (PL 044)

     a. TEMPERATURE RANGE*
        Blank = Commercial (0 to +70°C)
            I = Industrial (-40 to +85°C)
```
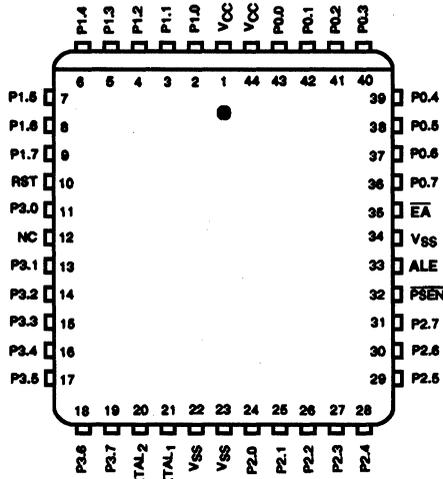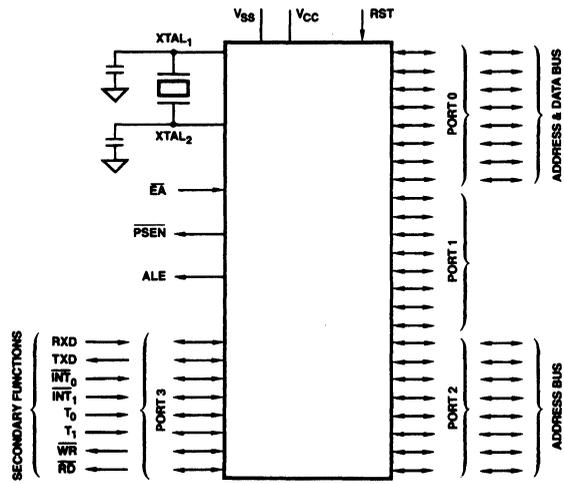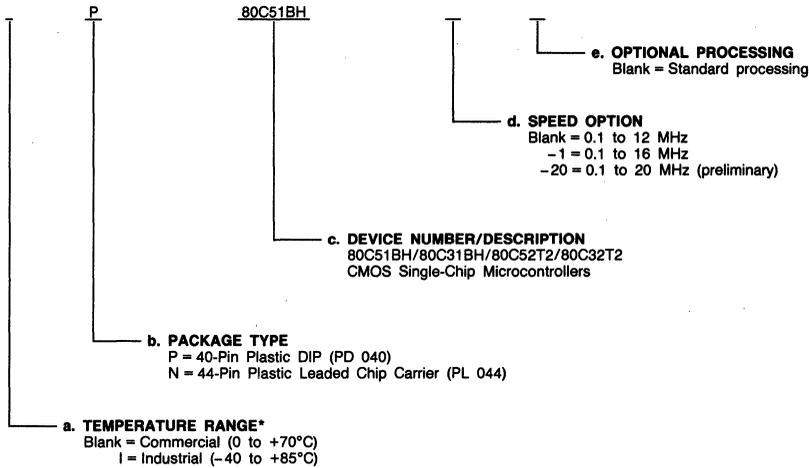
Valid Combinations

Valid Combinations	
P, N IP, IN	80C51BH
	80C51BH-1
	80C31BH
	80C31BH-1
P	80C31BH-20
P, N IP, IN	80C52T2-1
	80C32T2-1

*This device will also be available in Military temperature range.

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

Port 0   (Bidirectional, Open Drain)

Port 0 is an open-drain bidirectional I/O port. Port 0 pins that have 1s written to them float, and in that state can allow them to be used as high-impedance inputs.

Port 0 is also the multiplexed Low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also outputs the code bytes during program verification in the 80C51BH. External pullups are required during program verification.

Port 1   (Bidirectional)

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four LS TTL inputs. Port 1 pins that have 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 1 pins that are externally being pulled Low will source current (I_{IL} on the data sheet) because of the internal pullups.

Port 1 also receives the Low-order address bytes during program verification.

Port 2   (Bidirectional)

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four LS TTL inputs. Port 2 pins having 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 2 pins externally being pulled Low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the High-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function register.

Port 2 also receives the High-order address bits during ROM verification.

Port 3   (Bidirectional)

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four LS TTL inputs. Port 3 pins that have 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 3 pins externally being pulled Low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features as listed below:

Port Pin	Alternate Function
$P_{3.0}$	RxD (serial input port)
$P_{3.1}$	TxD (serial output port)
$P_{3.2}$	$\overline{INT_0}$ (external interrupt 0)
$P_{3.3}$	$\overline{INT_1}$ (external interrupt 1)
$P_{3.4}$	T_0 (Timer 0 external input)
$P_{3.5}$	T_1 (Timer 1 external input)
$P_{3.6}$	$\overline{WR}$ (external Data Memory write strobe)
$P_{3.7}$	$\overline{RD}$ (external Data Memory read strobe)

RST   Reset (Input, Active High)

A High on this pin (for two machine cycles while the oscillator is running) resets the device. An internal diffused resistor to V_{SS} permits power-on reset, using only an external capacitor to V_{CC}.

ALE   Address Latch Enable (Output, Active High)

Address Latch Enable is the output pulse for latching the Low byte of the address during accesses to external memory.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, allowing use for external-timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

$\overline{PSEN}$ Program Store Enable (Output, Active Low)

$\overline{PSEN}$ is the read strobe to external Program Memory. When the 80C51BH is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle—except that two $\overline{PSEN}$ activations are skipped during each access to external Data Memory. $\overline{PSEN}$ is not activated during fetches from internal Program Memory.

$\overline{EA}$ External Access Enable (Input, Active Low)

$\overline{EA}$ must be externally held Low to enable the device to fetch code from external Program Memory locations 0000H to 0FFFH. If $\overline{EA}$ is held High, the device executes from internal Program Memory unless the program counter contains an address greater than 0FFFH.

$XTAL_1$ Crystal (Input)

Input to the inverting-oscillator amplifier, and input to the internal clock-generator circuits.

$XTAL_2$ Crystal (Output)

Output from the inverting-oscillator amplifier.

V_{CC} Power Supply

Supply voltage during normal, idle, and power-down operations.

V_{SS} Circuit Ground

FUNCTIONAL DESCRIPTION

Oscillator Characteristics

XTAL$_1$ and XTAL$_2$ are the input and output, respectively, of an inverting amplifier which is configured for use as an on-chip oscillator (see Figure 1). Either a quartz crystal or ceramic resonator may be used.

To drive the device from an external clock source, XTAL$_1$ should be driven while XTAL$_2$ is left unconnected (see Figure 2). There are no requirements on the duty cycle of the external-clock signal since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum High and Low times specified on the data sheet must be observed.
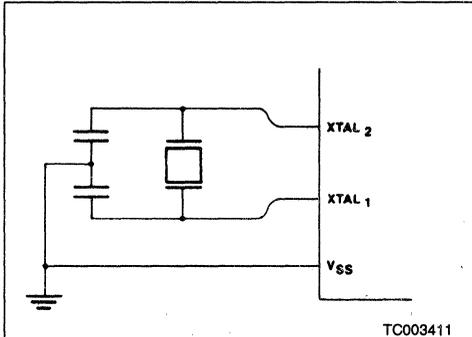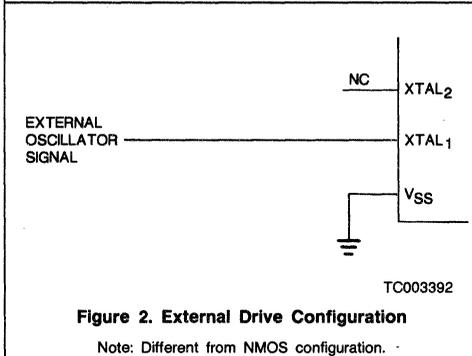

Figure 1. Crystal Oscillator

Figure 2. External Drive Configuration

Note: Different from NMOS configuration.

Idle and Power-Down Operation

Figure 3 shows the internal Idle and Power-Down clock configuration. As illustrated, Power-Down operation freezes the oscillator. Idle mode operation shows the interrupt, serial port, and timer blocks to continue to function while the clock to the CPU is halted.

These special modes are activated by software via the Special Function Register, PCON (Table 1). Its hardware address is 87H; PCON is not bit-addressable.

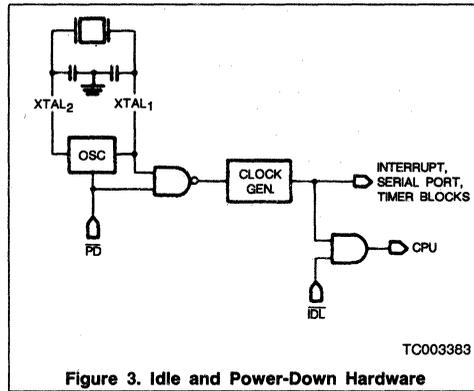If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is "0XXX0000."

Figure 3. Idle and Power-Down Hardware

TABLE 1. PCON (Power Control Register)

(MSB)							(LSB)
SMOD	–	–	–	GF1	GF0	PD	IDL

Symbol	Position	Name and Description
SMOD	PCON.7	Double-baud-rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2, or 3.
–	PCON.6	(Reserved)
–	PCON.5	(Reserved)
–	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit
GF0	PCON.2	General-purpose flag bit
PD	PCON.1	Power-Down bit. Setting this bit activates power-down operation.
IDL	PCON.0	Idle-mode bit. Setting this bit activates idle-mode operation.

Idle Mode

The instruction that sets PCON.0 is the last instruction executed in the normal operating mode before Idle mode is activated. Once in the Idle mode, the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all other registers maintain their data during Idle. Table 2 describes the status of the external pins during Idle mode.

There are two ways to terminate the Idle mode. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating Idle mode. The interrupt is serviced, and following RETI, the next instruction to be executed will be the one following the instruction that wrote a 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an enabled interrupt, the service routine can examine the status of the flag bits.

The second way of terminating the Idle mode is with a hardware reset. Since the oscillator is still running, the

hardware reset needs to be active for only 2 machine cycles (24 oscillator periods) to complete the reset operation.

Power-Down Mode

The instruction that sets PCON.1 is the last executed prior to going into Power-Down. Once in Power-Down, the oscillator is stopped. Only the contents of the on-chip RAM are preserved. The Special Function Registers are not saved. A hardware reset is the only way of exiting the Power-Down mode.

In the Power-Down mode, V_{CC} may be lowered to minimize circuit power consumption. Care must be taken to ensure the voltage is not reduced until the Power-Down mode is entered, and that the voltage is restored before the hardware reset is applied, which frees the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

Table 2 describes the status of the external pins while in the Power-Down mode. It should be noted that if the Power-Down mode is activated while in external program memory, the port data that is held in the Special Function Register P_2 is restored to Port 2. If the data is a 1, the port pin is held High during the Power-Down mode by the strong pullup, P_1, shown in Figure 4.

80C51BH I/O Ports

The I/O port drive of the 80C51BH is similar to the 8051. The I/O buffers for Ports 1, 2, and 3 are implemented as shown in Figure 4.
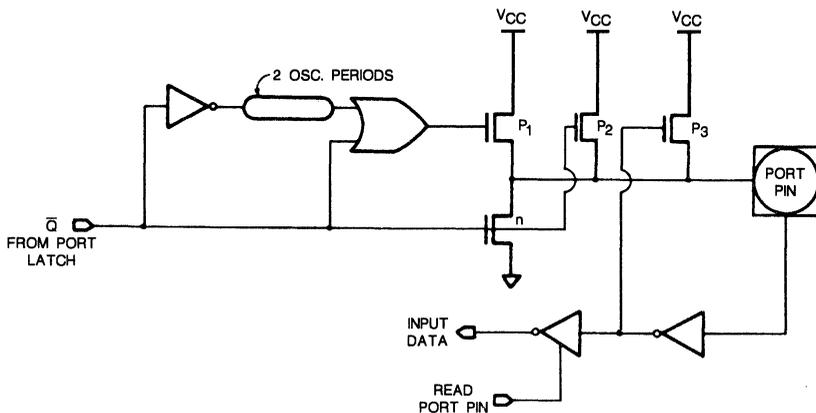
When the port latch contains a 0, all pFETS in Figure 4 are off while the nFET is turned on. When the port latch makes a 0-to-1 transition, the nFET turns off. The strong pullup pFET, P_1, turns on for two oscillator periods, pulling the output High very rapidly. As the output line is drawn High, pFET P_3 turns on through the inverter to supply the I_{OH} source current. This inverter and P_3 form a latch which holds the 1 and is supported by P_2.

When Port 2 is used as an address port, for access to external program of data memory, any address bit that contains a 1 will have its strong pullup turned on for the entire duration of the external memory access.

When an I/O pin on Ports 1, 2, or 3 is used as an input, the user should be aware that the external circuit must sink current during the logical 1-to-0 transition. The maximum sink current is specified as I_{TL} under the D.C. Specifications. When the input goes below approximately 2 V, P_3 turns off to save I_{CC} current. Note, when returning to a logical 1, P_2 is the only internal pullup that is on. This will result in a slow rise time if the user's circuit does not force the input line High.

TABLE 2. STATUS OF THE EXTERNAL PINS DURING IDLE AND POWER-DOWN MODES

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Port Data	Port Data	Port Data	Port Data
Idle	External	1	1	Floating	Port Data	Address	Port Data
Power-Down	Internal	0	0	Port Data	Port Data	Port Data	Port Data
Power-Down	External	0	0	Floating	Port Data	Port Data	Port Data

TC003402

Figure 4. I/O Buffers in the 80C51BH (Ports 1, 2, 3)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature ....–65°C to +150°C
Voltage on Any
Pin to V$_{SS}$ .... –0.5 V to V$_{CC}$ +0.5 V
Voltage on V$_{CC}$ to V$_{SS}$ .... –0.5 V to 6.5 V
Power Dissipation ....200 mW

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
  Temperature (T$_A$)...................................0 to +70°C
  80C51BH/80C31BH
    Supply Voltage (V$_{CC}$).......................+4 V to +6 V
  80C52T2/80C32T2
    Supply Voltage (V$_{CC}$)................. +4.5 V to +5.5 V
  Ground (V$_{SS}$)..0 V
Industrial (I) Devices
  Temperature (T$_A$)...............................–40 to +85°C
  80C51BH/80C31BH
    Supply Voltage (V$_{CC}$)................. +4.5 V to +5.5 V
  80C52T2/80C32T2
    Supply Voltage (V$_{CC}$)................. +4.5 V to +5.5 V
  Ground (V$_{SS}$)...0 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating ranges unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Units
V$_{IL}$	Input Low Voltage (Except $\overline{EA}$)		–0.5	.2 V$_{CC}$ – 0.1	V
V$_{IL1}$	Input Low Voltage ($\overline{EA}$)		–0.5	.2 V$_{CC}$ – 0.3	V
V$_{IH}$	Input High Voltage (Except XTAL$_1$, RST)		0.2 V$_{CC}$ + 0.9	V$_{CC}$ + 0.5	V
V$_{IH1}$	Input High Voltage (XTAL$_1$ RST)		0.7 V$_{CC}$	V$_{CC}$ + 0.5	V
V$_{OL}$	Output Low Voltage (Ports 1, 2, 3)	I$_{OL}$ = 1.6 mA (Note 1)		0.45	V
V$_{OL1}$	Output Low Voltage (Port 0, ALE, $\overline{PSEN}$)	I$_{OL}$ = 3.2 mA (Note 1)		0.45	V
V$_{OH}$	Output High Voltage (Ports 1, 2, 3)	I$_{OH}$ = –60 μA, V$_{CC}$ = 5 V±10%	2.4		V
		I$_{OH}$ = –25 μA	0.75 V$_{CC}$		V
		I$_{OH}$ = –10 μA	0.9 V$_{CC}$		V
V$_{OH1}$	Output High Voltage (Port 0 in External Bus Mode, ALE, $\overline{PSEN}$)	I$_{OH}$ = –400 μA, V$_{CC}$ = 5 V±10%	2.4		V
		I$_{OH}$ = –150 μA	0.75 V$_{CC}$		V
		I$_{OH}$ = –40 μA (Note 2)	0.9 V$_{CC}$		V
I$_{IL}$	Logical 0 Input Current (Ports 1, 2, 3)	V$_{IN}$ = 0.45 V		–50	μA
I$_{TL}$	Logical 1 to 0 Transition Current (Ports 1, 2, 3)	V$_{IN}$ = 2 V		–650	μA
I$_{LI}$	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V$_{IN}$ < V$_{CC}$		±10	μA
RRST	Reset Pulldown Resistor		50	150	kΩ
CIO	Pin Capacitance	Test Freq. = 1 MHz, T$_A$ = 25°C		10	pF
I$_{PD}$	Power Down Current	V$_{CC}$ = 2 to 6 V (Note 3)		50	μA

80C51BH/80C31BH MAXIMUM I$_{CC}$ (mA)

	Operating (Note 4)			Idle (Note 5)		
Freq. V$_{CC}$	4 V	5 V	6 V	4 V	5 V	6 V
0.1 MHz	1.2	1.5	2.5	0.5	0.7	1.1
3.5 MHz	4.3	5.7	7.5	1.1	1.6	2.2
8.0 MHz	8.3	11	14	1.8	2.7	3.7
12 MHz	12	16	20	2.5	3.7	5
16 MHz	16	20.5	25	3.5	5	6.5

80C52T2/80C32T2 MAXIMUM I$_{CC}$ (mA)

|	Operating (Note 4)			Idle (Note 5)		
Freq. V$_{CC}$	4.5 V	5.0 V	5.5 V	4.5 V	5.0 V	5.5 V
0.1 MHz	2.2	3.1	3.8	0.7	0.9	1.4
3.5 MHz	6	8	10	1.5	2	3
8.0 MHz	11	14	18	2.5	3.5	5
12 MHz	15	20	25	3.5	5	6
16 MHz	19	25	32	4.5	6.5	8.5

Notes: 1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V$_{OL}$S of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8 V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt-Trigger STROBE input.
2. Capacitive loading on Ports 0 and 2 may cause the V$_{OH}$ on ALE and $\overline{PSEN}$ to momentarily fall before the .9 V$_{CC}$ specification when the address bits are stabilizing.
3. Power-Down I$_{CC}$ is measured with all outputs pins disconnected; $\overline{EA}$ = Port 0 = V$_{CC}$; XTAL$_2$ N.C.; RST = V$_{SS}$.
4. I$_{CC}$ is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V$_{IL}$ = V$_{SS}$ + 0.5 V, V$_{IH}$ = V$_{CC}$ – 0.5 V; XTAL$_2$ NC; $\overline{EA}$ = RST = Port 0 = V$_{CC}$.
   I$_{CC}$ would be slightly higher if a crystal oscillator is used.
5. Idle I$_{CC}$ is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V$_{IL}$ = V$_{SS}$ + 0.5 V, V$_{IH}$ = V$_{CC}$ – 0.5 V; XTAL$_2$ NC; Port 0 = V$_{CC}$; $\overline{EA}$ = RST = V$_{SS}$.

SWITCHING CHARACTERISTICS over operating ranges unless otherwise specified

(C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF; C_L for All Other Outputs = 80 pF)
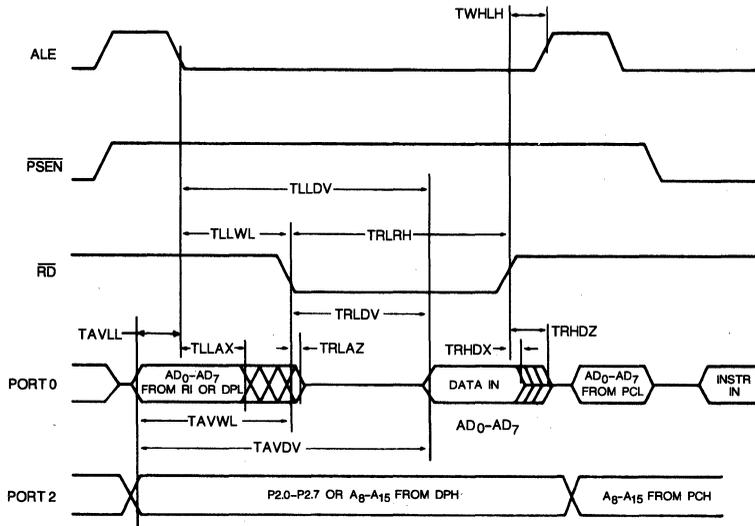
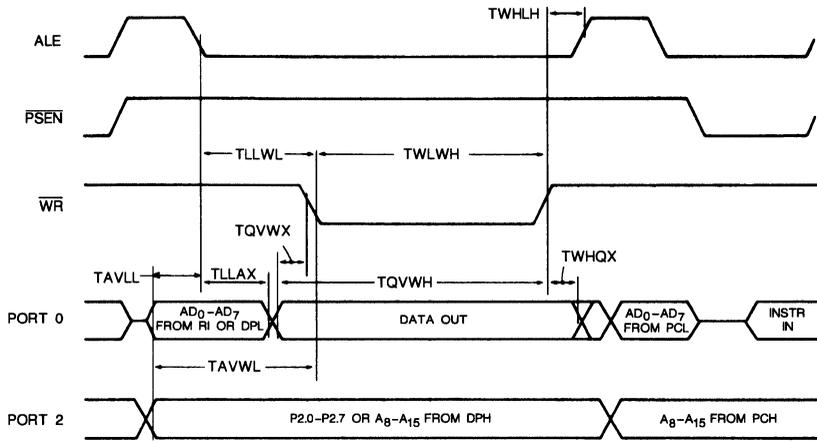| Parameter Symbol	Parameter Description	20 MHz Min	20 MHz Max	16-MHz Osc. Min	16-MHz Osc. Max	12-MHz Osc. Min	12-MHz Osc. Max	Variable Oscillator Min	Variable Oscillator Max	Units
External Program and Data Memory Characteristics										
1/TCLCL	Oscillator Frequency	0.1	20	0.1	16	0.1	12	0.1	16	MHz
TLHLL	ALE Pulse Width	60		85		127		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	20		7		28		TCLCL−55		ns
TLLAX	Address Hold After ALE Low	15		27		48		TCLCL−35		ns
TLLIV	ALE Low to Valid Instr. In		120		150		234		4TCLCL−100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	25		22		43		TCLCL−40		ns
TPLPH	$\overline{PSEN}$ Pulse Width	115		142		205		3TCLCL−45		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr. In		75		83		145		3TCLCL−105	ns
TPXIX	Input Instr. Hold After $\overline{PSEN}$	0		0		0		0		ns
TPXIZ	Input Instr. Float After $\overline{PSEN}$		35		38		59		TCLCL−25	ns
TAVIV	Address to Valid Instr. In		165		208		312		5TCLCL−105	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		0		10		10		10	ns
TRLRH	$\overline{RD}$ Pulse Width	200		275		400		6TCLCL−100		ns
TWLWH	$\overline{WR}$ Pulse Width	200		275		400		6TCLCL−100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		145		148		252		5TCLCL−165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		0		0		ns
TRHDZ	Data Float After $\overline{RD}$		60		55		97		2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		310		350		517		8TCLCL−150	ns
TAVDV	Address to Valid Data In		350		398		585		9TCLCL−165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	100	200	137	238	200	300	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address Valid to Read or Write Low	110		120		203		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	95		2		23		TCLCL−60		ns
TQVWH	Data Valid to Write High	200		287		433		7TCLCL−150		ns
TWHQX	Data Hold After $\overline{WR}$	25		12		33		TCLCL−50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	20	70	22	103	43	123	TCLCL−40	TCLCL+40	ns
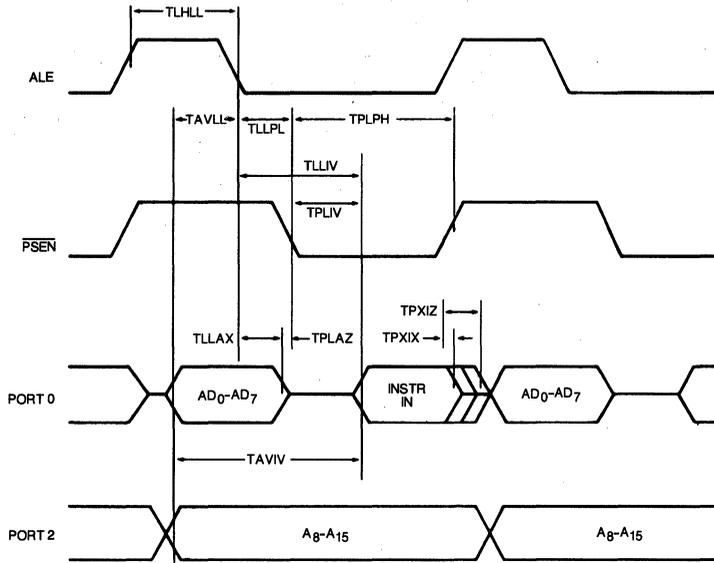
SWITCHING WAVEFORMS

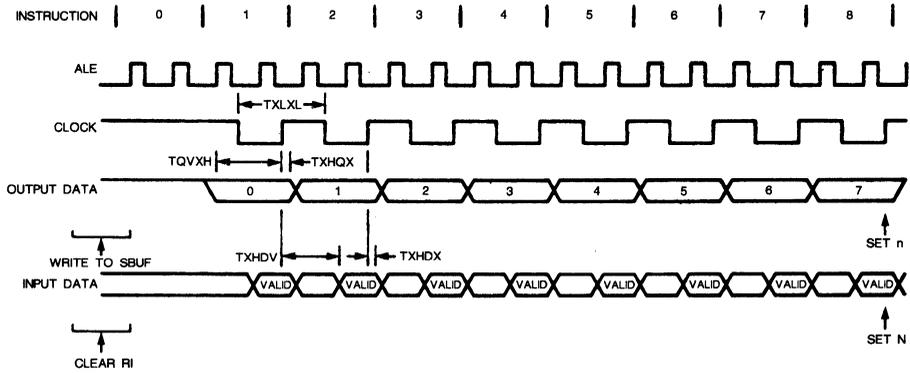**External Program Memory Read Cycle**

WF021962

External Data Memory Read Cycle

WF020962

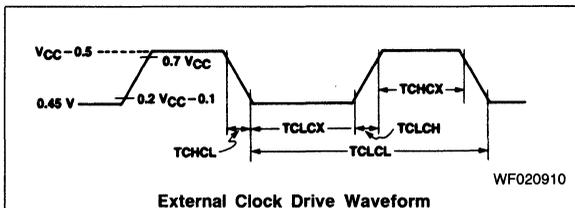# SWITCHING WAVEFORMS (continued)

External Data Memory Write Cycle

WF020932

Shift Register Timing Waveforms

WF020951

EXTERNAL CLOCK DRIVE

Parameter Symbol	Parameter Description	Min.	Max.	Units
1/TCLCL	Oscillator Frequency	0.1	20	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

WF020910

External Clock Drive Waveform

SERIAL PORT TIMING — SHIFT REGISTER MODE

Test Conditions: $T_A = 0°C$ to $70°C$; $V_{CC} = 5$ V $\pm 20\%$; $V_{SS} = 0$ V; Load Capacitance = 80 pF

Parameter Symbol	Parameter Description	16 MHz Osc.		Variable Oscillator		Units
		Min.	Max.	Min.	Max.	
TXLXL	Serial Port Clock Cycle Time	750		12TCLCL		ns
TQVXH	Output Data Setup to Clock Rising Edge	492		10TCLCL – 133		ns
TXHQX	Output Data Hold After Clock Rising Edge	8		2TCLCL – 117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		492		10TCLCL – 133	ns

WF020901

AC inputs during testing are driven at V_{CC} – 0.5 for a logic 1 and 0.45 V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

AC Testing Input/Output Waveforms
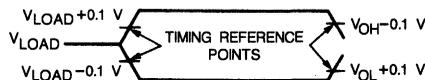
WF020941

For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, and begins to float when a 100 mV chance from the loaded V_{OH}/V_{OL} level occurs. $I_{OL}/I_{OH} \geq \pm 20$ mA.

Float Waveform

87C51/87C52T2

8-Bit CMOS Microcontrollers

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- Software- and pin-compatible with 80C51
- Beneficial for prototyping and initial production
- All 80C51BH and 80C52T2 features retained
- Flashrite™ EPROM programming
- Two-level Program Memory Lock
- 32-Byte Encryption Array
- In-Circuit Test Mode facilitates testing

	RAM (bytes)	EPROM (bytes)
87C51	128	4K
87C52T2	256	8K

87C51 = User-programmable 80C51BH
87C52T2 = User-programmable 80C52T2

GENERAL DESCRIPTION

The 87C51 and 87C52T2 are CMOS EPROM versions of the 80C51BH and 80C52T2, respectively. The 87C51 includes 4K bytes of on-chip EPROM, and the 87C52T2 includes 8K bytes of EPROM.

These user-programmable products are software- and pin-compatible with their ROM-based counterparts. All of the 80C52BH and 80C52T2 features are retained. For more information consult the 80C51/80C31BH/80C52T2/80C32T2 data sheet (order #04815).

Additionally, several new features are offered on the EPROM versions. The 87C51 and 87C52T2 EPROM array support the Flashrite programming algorithm that allows a 4K-byte EPROM array to be programmed in approximately 12 seconds. A two-level programmable lock structure prevents externally fetched code from accessing internal Program Memory and can disable EPROM verification and programming. A 32-byte Encryption Array can be used to encode the program code bytes during EPROM verification.

BLOCK DIAGRAM
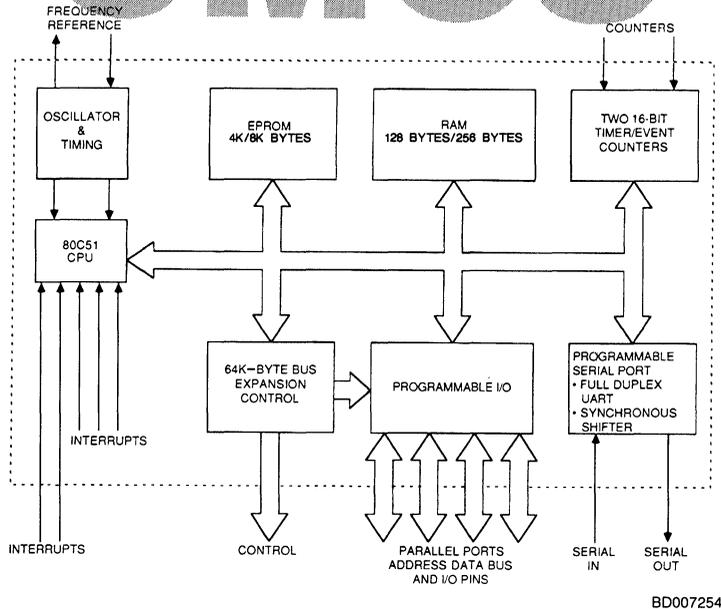
BD007254

CONNECTION DIAGRAMS
Top View

DIP
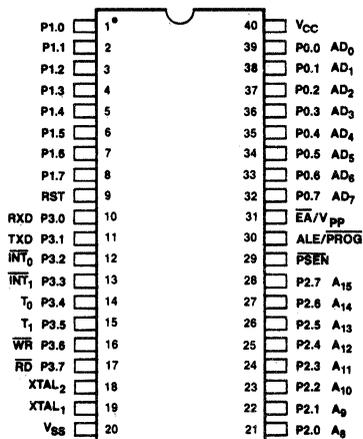
CD005553

LCC
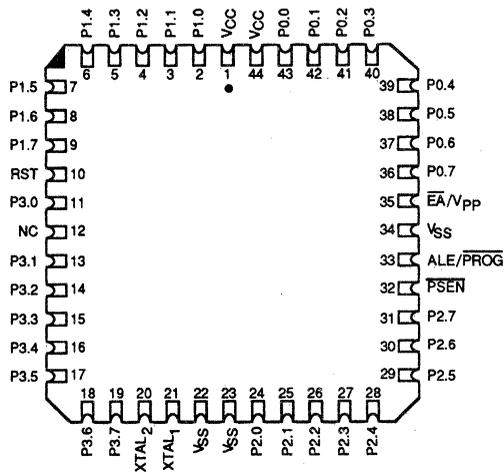
CD010873

PLCC
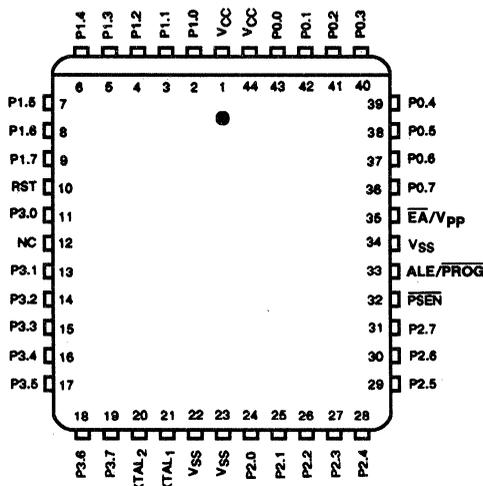
CD009442

Notes: Pin 1 is marked for orientation.

NC pins on the PLCC and LCC packages have been utilized as additional V_{CC} and V_{SS} connections to improve noise immunity. It is recommended that these pins (1, 23, and 37) be connected appropriately; however, they may be left floating to insure second source compatibility.

LOGIC SYMBOL
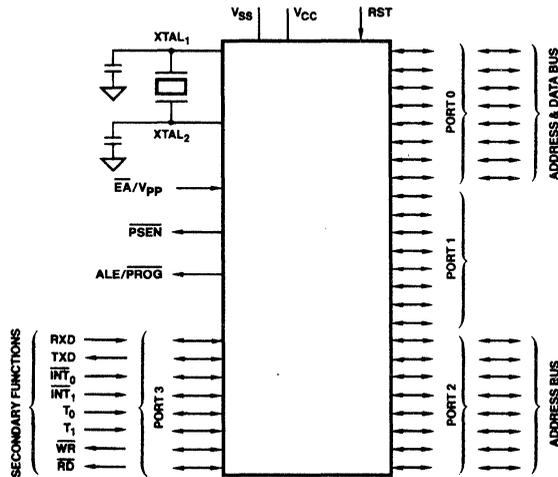
LS001326

ORDERING INFORMATION

Commodity Products

AMD commodity products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

a. **Temperature Range**
b. **Package Type**
c. **Device Number**
d. **Speed Option**
e. **Optional Processing**

```
    D      87C51
```

e. OPTIONAL PROCESSING
Blank = Standard processing

d. SPEED OPTION
Blank = 3.5 to 12 MHz
−1 = 3.5 to 16 MHz

c. DEVICE NUMBER/DESCRIPTION
87C51/87C52T2
8-Bit CMOS Microcontrollers
87C51 — 4K EPROM
87C52T2 — 8K EPROM

b. PACKAGE TYPE
D = 40-Pin Ceramic DIP (CDV040)
R = 44-Pin Ceramic Leadless Chip Carrier (CLV044)
P = 40-Pin Plastic DIP (PD 040)
N = 44-Pin Plastic Leadless Chip Carrier (PL 044)

a. TEMPERATURE RANGE
Blank = Commercial (0 to +70°C)
I = Industrial (−40 to +85°C)
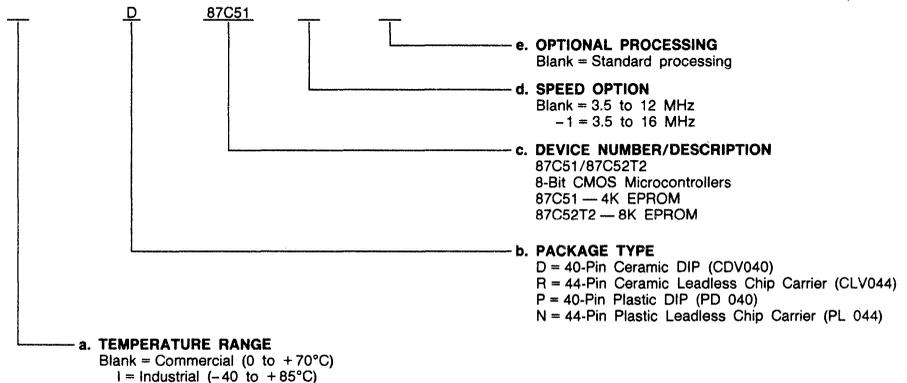
Valid Combinations	
D, R, P, N ID, IR, IP, IN	87C51
	87C51-1
	87C52T2
	87C52T2-1

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

Port 0   (Bidirectional; Open Drain)

Port 0 is an open-drain I/O port. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also outputs the code bytes during program verification in the 87C51/87C52T2. External pullups are required during program verification.

Port 1   (Bidirectional)

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four LS TTL inputs. Port 1 pins that have 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 1 pins that are externally being pulled Low will source current (I_{IL} on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification.

Port 2   (Bidirectional)

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four LS TTL inputs. Port 2 pins having 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 2 pins externally being pulled Low will source current (I_{IL}) because of internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function register.

Port 2 also receives the high-order address bits during the programming of the EPROM and during program verification of the EPROM, as well as some control signals.

Port 3   (Bidirectional)

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four LS TTL inputs. Port 3 pins having 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 3 pins externally being pulled Low will source current (I_{IL}) because of the pullups. Port 3 also receives some control signals for EPROM programming and program verification.

Port 3 also serves the functions of various special features as listed below:

Port Pin	Alternate Function
$P_{3.0}$	RxD (Serial Input Port)
$P_{3.1}$	TxD (Serial Output Port)
$P_{3.2}$	$\overline{INT_0}$ (External Interrupt 0)
$P_{3.3}$	$\overline{INT_1}$ (External Interrupt 1)
$P_{3.4}$	T_0 (Timer 0 External Input)
$P_{3.5}$	T_1 (Timer 1 External Input)
$P_{3.6}$	$\overline{WR}$ (External Data Memory Write Strobe)
$P_{3.7}$	$\overline{RD}$ (External Data Memory Read Strobe)

RST   Reset (Input; Active High)

This pin is used to reset the device when held High for two machine cycles while the oscillator is running. A small internal resistor permits power-on reset using only a capacitor connected to V_{CC}.

ALE/$\overline{PROG}$ Address Latch Enable/Program Pulse (Input/Output)

Address Latch Enable is the output pulse for latching the low byte of the address during accesses to external memory. ALE can drive eight LS TTL inputs.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, allowing use for external-timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory. This pin also accepts the program pulse input ($\overline{PROG}$) when programming the EPROM.

$\overline{PSEN}$ Program Store Enable (Output; Active Low)

$\overline{PSEN}$ is the read strobe to external Program Memory. $\overline{PSEN}$ can drive eight LS TTL inputs. When the device is executing code from an external program memory, $\overline{PSEN}$ is activated twice each machine cycle—except that two $\overline{PSEN}$ activations are skipped during each access to external Data Memory. $\overline{PSEN}$ is not activated during fetches from internal Program Memory.

$\overline{EA}$/V_{PP} External Access Enable/Programming Voltage (Input; Active Low)

$\overline{EA}$ must be externally held Low to enable the device to fetch code from external Program Memory locations 0000H to 0FFFH. If $\overline{EA}$ is held High, the 87C51/87C52T2 executes from internal Program Memory unless the program counter contains an address greater than 0FFFH.

This pin also receives the 12.75-V programming supply voltage during programming of the EPROM.

XTAL₁   Crystal (Input)

Input to the inverting-oscillator amplifier, and input to the internal clock-generator circuits.

XTAL₂   Crystal (Output)

Output of the inverting-oscillator amplifier.

V_CC   Power Supply

Power supply during normal, idle, and power-down operations.

V_SS   Circuit Ground

PROGRAMMING

The 87C51/87C52T2 can be programmed with the Flashrite algorithm. It differs from other methods in the value used for V_{PP} (programming supply voltage) and in the width and number of the ALE/$\overline{PROG}$ pulses.

To program the EPROM, either the internal or external oscillator must be running between 4 and 6 MHz, since the internal bus is used to transfer address and program data to the appropriate internal registers. Table 1 shows the various EPROM programming modes.

TABLE 1. EPROM PROGRAMMING MODES FOR THE 87C51/87C52T2

Mode	RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	$\overline{EA}/V_{PP}$	P2.7	P2.6	P3.7	P3.6
Program Code	H	L	L*	V_{PP}	H	L	H	H
Verify Code	H	L	H	V_{PPX}	L	L	H	H
Pgm Encryption Table	H	L	L*	V_{PP}	H	L	H	L
Pgm Lock Bit 1	H	L	L*	V_{PP}	H	H	H	H
Pgm Lock Bit 2	H	L	L*	V_{PP}	H	H	L	L
Read Silicon Signature	H	L	H	H	L	L	L	L

Key: H = Logic High for that pin
L = Logic Low for that pin
V_{PP} = 12.75 V ±0.25 V
V_{CC} = 5 V ±10% during programming and verification
2.0 V < V_{PPX} < 13.0 V

*ALE/$\overline{PROG}$ receives 25 programming pulses while V_{PP} is held at 12.75 V. Each programming pulse is low for 100 μs (±10% μs) and high for a minimum of 10 μs.

Programming

The programming configuration is shown in Figure 1. The address of the EPROM location to be programmed is applied to Ports 1 and 2 as shown in the figure. The programming configuration of the 87C52T2 is identical except that P2.4 is also used as an address input. The code byte to be programmed into that location is applied to Port 0. Once RST, $\overline{PSEN}$, Port 2, and Port 3 are held to the levels indicated in Figure 1, ALE/$\overline{PROG}$ is pulsed low 25 times as shown in Figure 2.

The maximum voltage applied to the $\overline{EA}/V_{PP}$ pin must not exceed 13 V at any time as specified for V_{PP}. Even a slight spike can cause permanent damage to the device. The V_{PP} source should thus be well regulated and glitch-free.

When programming, a 0.1 μF capacitor is required across V_{PP} and ground to suppress spurious transients which may damage the device.
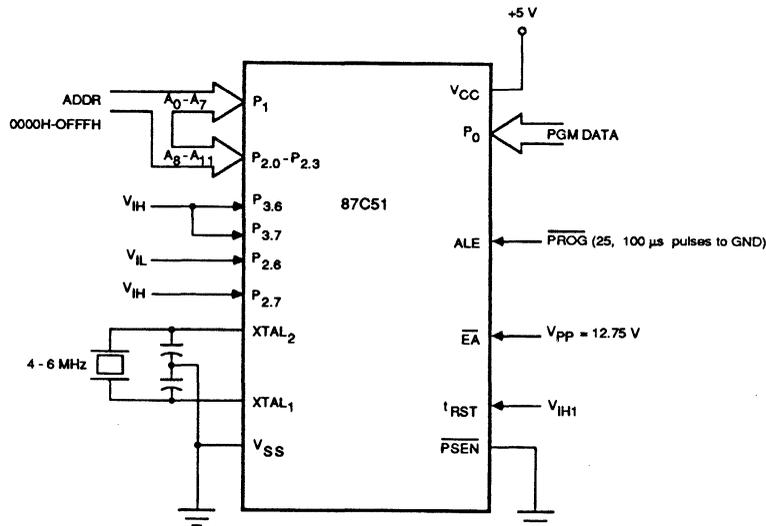
TC004691

Figure 1. 87C51 Programming Configuration
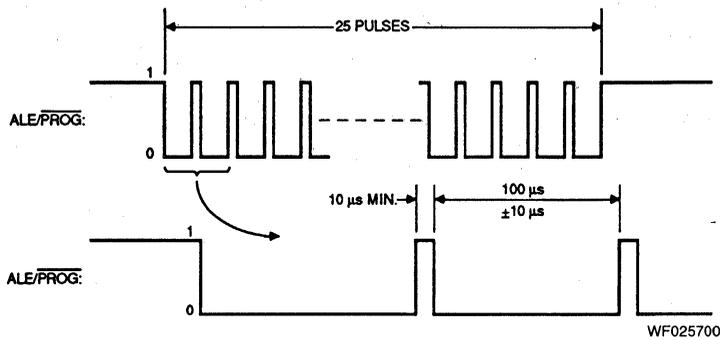
Figure 2. $\overline{PROG}$ Waveforms

Program Verification

The 87C51/87C52T2 provides a method of reading the programmed code bytes in the EPROM array for program verification. This function is possible as long as Lock Bit 2 has not been programmed.

For program verification, the address of the Program Memory location to be read is applied to Ports 1 and 2 as shown in

Figure 3. Verification of the 87C52T2 is identical except that P2.4 is also used as an address input. Once RST, $\overline{PSEN}$, Port 2, and Port 3 are held to the levels indicated, the contents of the addressed location will be emitted on Port 0. External pullups are required on Port 0 for this operation. The EPROM programming and verification waveforms provide further details.
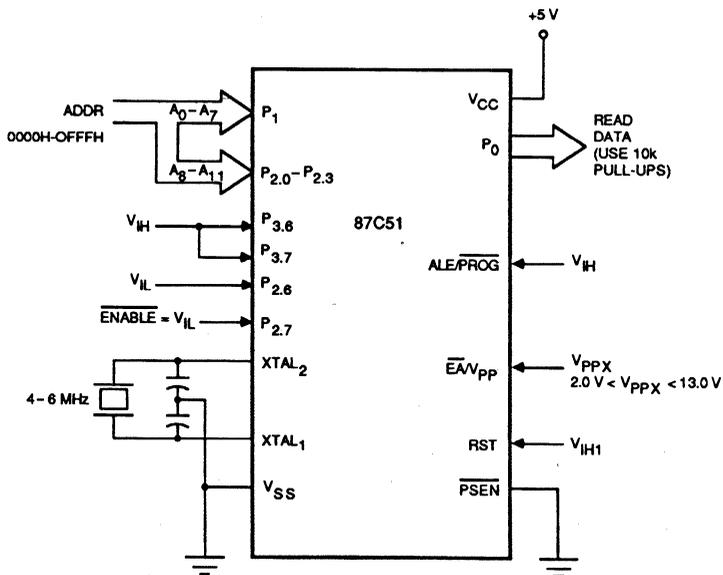
Figure 3. 87C51 Program Verification

Program Encryption Table

The 87C51/87C52T2 features a 32-byte Encryption Array. It can be programmed by the customer, thus encrypting the program code bytes read during EPROM verification. The EPROM verification procedure is performed as usual except that each code byte comes out logically X-NORed with one of the 32 key bytes.

The key byte used is the one whose address corresponds to the lower 5 bits of the EPROM verification address. Thus, when the EPROM is verified starting with address 0000H, all 32 keys in their correct sequence must be known. Unprogrammed bytes have the value FFH. Thus, if the Encryption Table is left unprogrammed, no encryption will be performed, since any byte X-NORed with FFH leaves that byte unchanged.

To program the Encryption Table, programming is set up as usual, except that P3.6 is held Low, as shown in Table 1. The 25-pulse programming sequence is applied to each address, 00 through 1FH. The programming of these bytes does not affect the standard 4K-byte EPROM array. When the Encryption Table is programmed, the Program Verify operation will produce only encrypted data.

The Encryption Table cannot be directly read. The programming of Lock Bit 1 will disable further Encryption Table programming.

Security Lock Bits

The 87C51 contains two Lock Bits which can be programmed to obtain additional security features. P = Programmed and U = Unprogrammed.

Lock Bit 1	Lock Bit 2	Result
U	U	Normal Operation
P	U	• Externally fetched code cannot access internal Program Memory • All further Programming disabled (except Lock Bit 2)
U	P	Reserved
P	P	• Externally fetched code cannot access internal Program Memory • All further Programming disabled • Program Verification disabled

To program the Lock Bits, a 100 pulse programming sequence is required using the levels shown in Table 1. After Lock Bit 1 is programmed, further programming of the Code Memory and Encryption Table is disabled. However, Lock Bit 2 may still be programmed, providing the highest level of security available on the 87C51/87C52T2.

When Lock Bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

Silicon Signature Verification

AMD supports silicon signature verification for the 87C51/87C52T2. The manufacturer code and part code can be read from the device before any programming is done to enable the EPROM Programmer to recognize the device.

To read the silicon signature, the external pins are set up as shown in Figure 4. This procedure is the same as a normal verification except that P3.6 and P3.7 are pulled to a logic Low. The values returned are:

Manufacturer Code	Address: 0030H	Code: 01H
87C51 Part Code	Address: 0031H	Code: B0H
87C52T2 Part Code	Address: 0031H	Code: 31H

Code 01H indicates AMD as the manufacturer. Code B0H indicates the device type is the 87C51, and Code 31H indicates a 87C52T2.

In-Circuit Test Mode

The In-Circuit Test Mode facilitates testing and debugging of systems using the 87C51 without the 87C51 having to be removed from the circuit. The In-Circuit Test Mode is invoked by:

1. Pulling ALE Low while RST is held High, and $\overline{PSEN}$ is High.

2. Holding ALE Low as RST is de-activated.

While the device is in In-Circuit Test Mode, the Port 0 pins go into a float state, and the other port pins and ALE and $\overline{PSEN}$ are weakly pulled High. The oscillator circuit remains active. While the 87C51 is in this mode, an emulator or test CPU can be used to drive the circuit. Normal operation is restored when a Hardware Reset is applied.

Erasure Characteristics

Light and other forms of electromagnetic radiation can lead to erasure of the EPROM when exposed for extended periods of time.

Wavelengths of light shorter than 4000 angstroms, such as sunlight or indoor fluorescent lighting, can ultimately cause inadvertent erasure and should, therefore, not be allowed to expose the EPROM for lengthy durations (approximately one week in sunlight or three years in room-level fluorescent lighting). It is suggested that the window be covered with an opaque label if an application is likely to subject the device to this type of radiation.

It is recommended that ultraviolet light (of 2537 angstroms) be used to a dose of at least 15 W-sec/cm^2 when erasing the EPROM. An ultraviolet lamp rated at 12,000 μW/cm^2 held one inch away for 20–30 minutes should be sufficient.

EPROM erasure leaves the Program Memory in an "all ones" state.
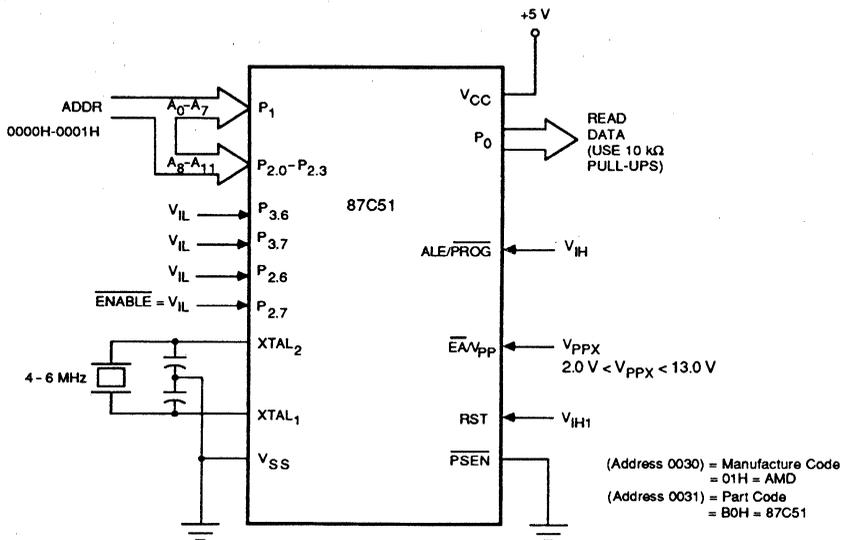
Figure 4. 87C51 Silicon Signature Verification Configuration

Oscillator Characteristics

XTAL$_1$ and XTAL$_2$ are the input and output, respectively, of an inverting amplifier which is configured for use as an on-chip oscillator (see Figure 5). Either a quartz crystal or ceramic resonator may be used.

To drive the device from an external clock source, XTAL$_1$ should be driven while XTAL$_2$ is left unconnected (see Figure 6). There are no requirements on the duty cycle of the external clock signal since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum High and Low times specified on the data sheet must be observed.

Figure 5. Crystal Oscillator

Figure 6. External Drive Configuration

ABSOLUTE MAXIMUM RATINGS

Storage Temperature .... −65 to +150°C
Voltage on $\overline{EA}$/V$_{PP}$ Pin to V$_{SS}$ ....−0.5 to +13.0 V
Voltage on V$_{CC}$ to V$_{SS}$ .... −0.5 to +6.5 V
Voltage on Any Other Pin to V$_{SS}$ .... −0.5 to +6.5 V
Power Dissipation ....200 mW

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
   Ambient Temperature (T$_A$) .... 0 to +70°C
   Supply Voltage (V$_{CC}$) ....+4.5 to +5.5 V
   Ground (V$_{SS}$)..0 V
Industrial (I) Devices
   Ambient Temperature (T$_A$) ....−40 to +85°C
   Supply Voltage (V$_{CC}$) ....+4.5 to +5.5 V
   Ground (V$_{SS}$)..0 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating ranges

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
V$_{IL}$	Input Low Voltage (Except $\overline{EA}$)		−0.5	0.2 V$_{CC}$ − 0.1	V
V$_{IL1}$	Input Low Voltage ($\overline{EA}$)		0	0.2 V$_{CC}$ − 0.3	V
V$_{IH}$	Input High Voltage (Except XTAL$_1$, RST)		0.2 V$_{CC}$ + 0.9	V$_{CC}$ + 0.5	V
V$_{IH1}$	Input High Voltage to XTAL$_1$, RST		0.7 V$_{CC}$	V$_{CC}$ + 0.5	V
V$_{OL}$	Output Low Voltage (Ports 1, 2, 3)	I$_{OL}$ = 1.6 mA (Note 1)		0.45	V
V$_{OL1}$	Output Low Voltage (Port 0, ALE, $\overline{PSEN}$)	I$_{OL}$ = 3.2 mA (Note 1)		0.45	V
V$_{OH}$	Output High Voltage (Ports 1, 2, 3), ALE, $\overline{PSEN}$	I$_{OH}$ = −60 μA, V$_{CC}$ = 5 V ±10%	2.4		V
		I$_{OH}$ = −10 μA	0.9 V$_{CC}$		
V$_{OH1}$	Output High Voltage (Port 0 in External Bus Mode)	I$_{OH}$ = −800 μA, V$_{CC}$ = 5 V ±10%	2.4		V
		I$_{OH}$ = −80 μA (Note 2)			
I$_{IL}$	Logical 0 Input Current (Ports 1, 2, 3)	V$_{IN}$ = 0.45 V		−50	μA
I$_{TL}$	Logical 1-to-0 Transition Current (Ports 1, 2, 3)	(Note 3)		−650	μA
I$_{LI}$	Input Leakage Current (Port 0)	V$_{IN}$ = V$_{IL}$ or V$_{IH}$		±10	μA
I$_{CC}$	Power Supply Current: Active Mode @ 12 MHz (Note 4)	(Note 5)		Note 4	mA
	Idle Mode @ 12 MHz (Note 4)			Note 4	
	Power-Down Mode			50	μA
RRST	Reset Pulldown Resistor		50	300	kΩ
C$_{IO}$	Pin Capacitance	Test Freq = 1 MHz, T$_A$ = 25°C		10	pF

Notes: 1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V$_{OL}$s of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8 V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
2. Capacitive loading on Ports 0 and 2 may cause the V$_{OH}$ on ALE and $\overline{PSEN}$ to momentarily fall below the 0.9 V$_{CC}$ specification when the address bits are stabilizing.
3. Pins of Ports 1, 2, and 3 source a transition current when they are being externally driven from 1 to 0. The transition current reaches its maximum value when V$_{IN}$ is approximately 2 V.
4. I$_{CCMAX}$ at other frequencies is given by:
    Active Mode: I$_{CC}$ TYPICAL = 0.94 x Freq + 13.71   I$_{CC}$ MAX = 1.38 x Freq + 20.4
    Idle Mode: I$_{CC}$ TYPICAL = 0.38 x Freq + 5.4   I$_{CC}$ MAX = 0.38 x Freq + 11.9
    where Freq is the external oscillator frequency in MHz. I$_{CCMAX}$ is given in mA (see Figure 5).
5. Active Mode I$_{CC}$ is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V$_{IL}$ = V$_{SS}$ + 0.5 V, V$_{IH}$ = V$_{CC}$ − 0.5 V; XTAL$_2$ N.C.; $\overline{EA}$ = RST = Port 0 = V$_{CC}$.
    Idle Mode I$_{CC}$ is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V$_{IL}$ = V$_{SS}$ + 0.5 V, V$_{IH}$ = V$_{CC}$ − 0.5 V; XTAL$_2$ = N.C.; Port 0 = V$_{CC}$; $\overline{EA}$ = RST = V$_{SS}$.
    Power-Down Mode I$_{CC}$ is measured with all output pins disconnected; $\overline{EA}$ = Port 0 = V$_{CC}$; XTAL$_2$ NC; RST = V$_{SS}$.

SWITCHING CHARACTERISTICS over operating ranges
(Load Capacitance for Port 0, ALE, and $\overline{\text{PSEN}}$ = 100 pF, Load Capacitance for All Other Outputs = 80 pF)

Parameter Symbol	Parameter Description	16 MHz Osc.		12 MHz Osc.		Variable Oscillator		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
1/TCLCL	Oscillator Frequency					3.5	16	MHz
TLHLL	ALE Pulse Width	85		127		2TCLCL-40		ns
TAVLL	Address Valid to ALE Low	7		28		TCLCL-55		ns
TLLAX	Address Hold After ALE Low	27		48		TCLCL-35		ns
TLLIV	ALE Low to Valid Instr. In		150		234		4TCLCL-100	ns
TLLPL	ALE Low to $\overline{\text{PSEN}}$ Low	22		43		TCLCL-40		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	142		205		3TCLCL-45		ns
TPLIV	$\overline{\text{PSEN}}$ Low to Valid Instr. In		83		145		3TCLCL-105	ns
TPXIX	Input Instr. Hold After $\overline{\text{PSEN}}$	0		0		0		ns
TPXIZ	Input Instr. Float After $\overline{\text{PSEN}}$		38		59		TCLCL-25	ns
TAVIV	Address to Valid Instr. In		208		312		5TCLCL-105	ns
TPLAZ	$\overline{\text{PSEN}}$ Low to Address Float		10		10		10	ns
TRLRH	$\overline{\text{RD}}$ Pulse Width	275		400		6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	275		400		6TCLCL-100		ns
TRLDV	$\overline{\text{RD}}$ Low to Valid Data In		148		252		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{\text{RD}}$	0		0		0		ns
TRHDZ	Data Float After $\overline{\text{RD}}$		55		97		2TCLCL-70	ns
TLLDV	ALE Low to Valid Data In		350		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		398		585		9TCLCL-165	ns
TLLWL	ALE Low to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low	137	238	200	300	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address Valid to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low	120		203		4TCLCL-130		ns
TQVWX	Data Valid to $\overline{\text{WR}}$ Transition	2		23		TCLCL-60		ns
TQVWH	Data Valid to $\overline{\text{WR}}$ High	287		433		7TCLCL-150		ns
TWHQX	Data Hold After $\overline{\text{WR}}$	12		33		TCLCL-50		ns
TRLAZ	$\overline{\text{RD}}$ Low to Address Float		0		0		0	ns
TWHLH	$\overline{\text{RD}}$ or $\overline{\text{WR}}$ High to ALE High	22	103	43	123	TCLCL-40	TCLCL+40	ns

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

87C51/87C52T2

SWITCHING WAVEFORMS

External Program Memory Read Cycle

WF021962

External Data Memory Read Cycle

WF020962

SWITCHING WAVEFORMS (continued)

WF020932

External Data Memory Write Cycle

WF020951

Shift Register Timing Waveforms

87C51/87C52T2

EXTERNAL CLOCK DRIVE

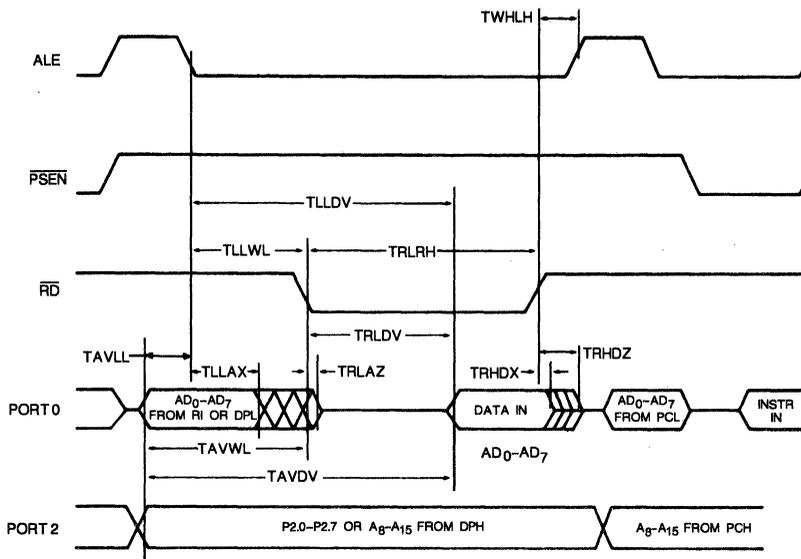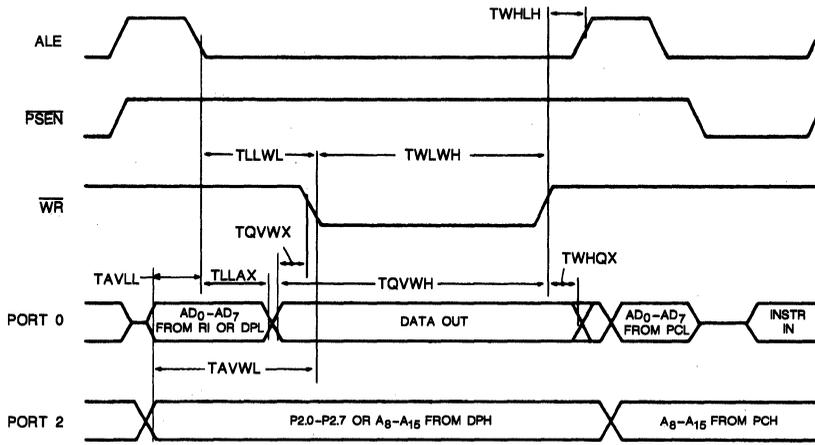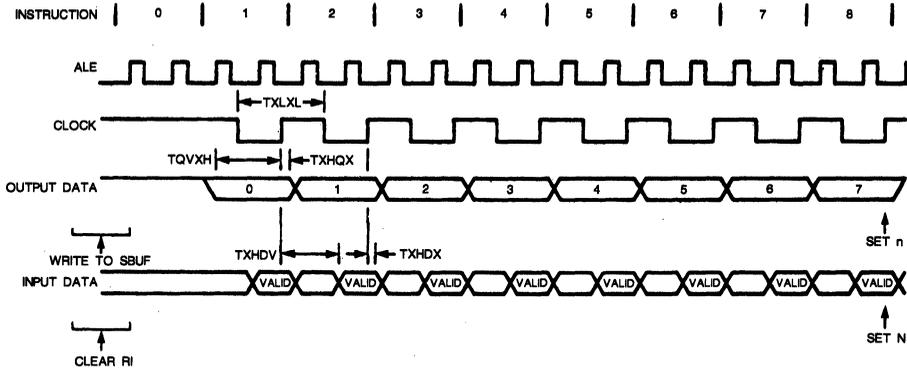| Parameter Symbol	Parameter Description	Min.	Max.	Unit
1/TCLCL	Oscillator Frequency	3.5	16	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

External Clock Drive Waveform

WF020910

SERIAL PORT TIMING — SHIFT REGISTER MODE
(Test Conditions: T_A = 0 to +70°C; V_{CC} = 5 V ±10%; V_{SS} = 0 V; Load Capacitance = 80 pF)

Parameter Symbol	Parameter Description	16 MHz Osc.		Variable Oscillator		Unit
		Min.	Max.	Min.	Max.	
TXLXL	Serial Port Clock Cycle Time	750		12TCLCL		ns
TQVXH	Output Data Setup to Clock Rising Edge	492		10TCLCL–133		ns
TXHQX	Output Data Hold After Clock Rising Edge	8		2TCLCL–117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		492		10TCLCL–133	ns

AC Testing

AC inputs during testing are driven at V_{CC} – 0.5 for a logic 1 and 0.45 V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

WF020901

Input/Output Waveform

For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, and begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs. $I_{OL}/I_{OH} \geq \pm 20$ mA.

WF020941

Float Waveform

EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS
$(T_A = +21$ to $+27°C)$

Parameter Symbol	Parameter Description	Min.	Max.	Unit
V$_{PP}$	Programming Supply Voltage	12.5	13.0	V
I$_{PP}$	Programming Supply Current		50	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to PROG	48TCLCL		
TGHAX	Address Hold After PROG	48TCLCL		
TDVGL	Data Setup to PROG	48TCLCL		
TGHDX	Data Hold After PROG	48TCLCL		
TEHSH	P2.7 (ENABLE) High to V$_{PP}$	48TCLCL		
TSHGL	V$_{PP}$ Setup to PROG	10		μs
TGHSL	V$_{PP}$ Hold after PROG	10		μs
TGLGH	PROG Width	90	110	μs
TAVQV	Address to Data Valid		48TCLCL	
TELQV	ENABLE to Data Valid		48TCLCL	
TEHQZ	Data Float After ENABLE	0	48TCLCL	
TGHGL	PROG High to PROG Low	10		μs

EPROM PROGRAMMING AND VERIFICATION WAVEFORMS

WF025693

For Programming conditions, see Figures 1 and 2.
For Verification conditions, see Figure 3.

Designing with the 80C51BH

CMOS EVOLVES

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are functional equivalents to the correspondingly numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well-known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed was due to the technology used to construct the transistors in the circuit.

This technology is called metal-gate CMOS, because the transistor gates are formed by metal deposition. More importantly, the gates are formed after the drain and source regions have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors result in high electrode capacitance; that is what limits the speed of the circuit.

High-speed CMOS became feasible with the development of the self-aligning silicon-gate technology. In this process, polysilicon gates are deposited **before** the source and drain regions are defined. Then the source and drain regions are formed by ion implantation using the gate itself as a mask for the implantation. This eliminates most of the overlap capacitance. In addition, the process allows smaller transistors, resulting in a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speed comparable to LS TTL, which is to say about 10 times faster than the 74C-series circuits.

The size reduction that contributes to the higher speed also demands an accompanying reduction in the maximum supply voltage. High-speed CMOS is generally limited to 6 V.

WHAT IS CMOS?

There are two CMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel transistors (nFETs) are built into the substrate and pFETs are built into the n-wells. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the nFETs are built into the wells and pFETs into the substrate. Both processes have advantages and disadvantages, which are largely unseen by the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with NMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded NMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistors in a CMOS chip to be constructed by processes that are already well characterized.

THE 8051 FAMILY IN CMOS

The 80C51BH is the CMOS version of the original 8051. The 80C31BH is the ROMless 80C51BH, equivalent to the 8031. These CMOS devices are architecturally identical with their NMOS counterparts, except that they have two added features for reduced power: Idle and Power-Down modes of operation.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the 80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less in the reduced power modes). Interchangeability between the NMOS and CMOS devices is discussed in more detail in the final section.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the node-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area to produce a more economical device.

This is not to say that the on-chip RAM in CMOS microcontrollers is dynamic. It's not. It is the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

LATCHUP

Latchup is an SCR-type turn-on phenomenon that is the traditional nemesis of CMOS systems. The substrate, the wells, and the transistors form parasitic pnpn structures within the device. These parasitic structures turn on like an SCR if a sufficient amount of forward current is driven through one of the junctions. From the circuit designer's point of view, it can happen whenever an input or output pin is externally driven a diode drop above V_{CC} or

below Vss by a source that is capable of supplying the required trigger current.

However much of a problem latchup has been in the past, it is good to know that in most recently developed CMOS devices, the current required to trigger latchup is typically well over 100 mA. The 80C51BH is virtually immune to latchup. (References 1 and 2 present a discussion of the latchup mechanisms and the steps that are taken on the chip to guard against it.) Modern CMOS is not immune to latchup, but with trigger currents in the hundreds of mA, latchup is certainly a lot easier to avoid than it once was.

A careless power-up sequence might trigger latchup in the older CMOS families, but it's unlikely to be a major problem in high-speed CMOS. There is still some risk incurred in inserting or removing chips or boards in a CMOS system while the power is on. Also, severe transients, such as inductive kicks or momentary short circuits, can exceed the trigger current for latchup.

For applications in which some latchup risk seems unavoidable, put a small resistor (100 ohms or so) in series with the signal lines to ensure that the trigger current will never be reached. This also helps to control overshoot and RFI.

LOGIC LEVELS AND INTERFACING PROBLEMS

CMOS logic levels differ from TTL levels in two ways. First, for equal supply voltages, CMOS gives (and requires) a higher "logic 1" level than TTL. Secondly, CMOS logic levels are Vcc (or VDD) dependent, whereas guaranteed TTL logic levels are fixed when Vcc is within TTL specs.

Standard 74HC logic levels are as follows:

V_{IH} min = 70% of Vcc

V_{IL} max = 20% of Vcc

V_{OH} min = Vcc − 0.1 V, $|I_{OH}| \leq 20\ \mu A$

V_{OL} max = 0.1V, $|I_{OL}| \leq 20\ \mu A$

Table 7-1 compares 74HC, LS TTL, and 74HCT logic levels with those of the NMOS 8051 and CMOS 80C51BH for Vcc = 5 V.

Output logic levels depend of course on load current, and are normally specified at several load currents. When CMOS and TTL are powered by the same Vcc, the logic levels guaranteed on the data sheets indicate that CMOS can drive TTL, but TTL cannot drive CMOS. The incompatibility is that the TTL circuit's VoH level is too low to reliably be recognized by the CMOS circuit as a valid VIH. Since NMOS circuits were designed to be TTL-compatible, they have the same incompatibility.

Fortunately, 74HCT-series circuits are available to ease these interfacing problems. They have TTL-compatible logic levels at the inputs and standard CMOS levels at the outputs.

The 80C51BH is designed to work with either TTL or CMOS. Therefore, its logic levels are specified very much like 74HCT circuits. That is, its input logic levels are TTL-compatible, and its output characteristics are like standard high-speed CMOS.

NOISE CONSIDERATIONS

One of the major reasons for going to CMOS has traditionally been that CMOS is less susceptible to noise than TTL. As previously noted, its low susceptibility to noise is partly due to superior noise margins, and partly due to its slow speed.

Noise margin is the difference between VoL and VIL, or between VoH and VIH. If VoH from a driving circuit is 2.7 V and VIH to the driven circuit is 2.0 V, then the driven circuit has 0.7 V of noise margin at the logic high level. These kinds of comparisons show that an all-CMOS system has wider noise margins than an all-TTL system. Table 7-2 shows noise margins in CMOS and LS TTL systems when both have Vcc = 5 V; CMOS/CMOS systems have an edge over LS TTL in this respect.

Table 7-1. Logic Level Comparison (output levels are for minimum loading)

Logic State	$V_{cc} = 5V$				
	74HC	74HCT	LS TTL	8051	80C51BH
V_{IH}	3.5 V	2.0 V	2.0 V	2.0 V	1.9 V
V_{IL}	1.0 V	0.8 V	0.8 V	0.8 V	0.9 V
V_{OH}	4.9 V	4.9 V	2.7 V	2.4 V	4.5 V
V_{OL}	0.1 V	0.1 V	0.5 V	0.45 V	0.45 V

Noise margins can be misleading, however, because they do not say how much noise energy in the circuit it takes to induce a noise voltage of sufficient amplitude to cause a logic error. This involves consideration of the width of the noise pulse as compared with the circuit response speed, and the impedance to ground from the point of noise introduction in the circuit.

Table 7-2. Noise Margins for CMOS and LS TTL Circuits

| Interface | Noise Margins for V_{cc} = 5 V | |
	Logic Low V_{IL}–V_{OL}	Logic High V_{OH}–V_{IH}
74HC to 74HC	0.9 V	1.4 V
LSTTL to LSTTL	0.3 V	0.7 V
LSTTL to 74HCT	0.3 V	0.7 V
LSTTL to 80C51BH	0.3 V	0.7 V
74HC to 80C51BH	0.8 V	3.0 V
80C51BH to 74HC	0.8 V	1.0 V

When these considerations are included, it is seen that using the slower 74C- and 4000-series circuits with a 12- or 15-V supply voltage does offer a truly improved level of noise immunity, but that high-speed CMOS at 5 V is not significantly better than TTL.

One should not mistake the wider supply voltage tolerance of high-speed CMOS for V_{cc} glitch immunity. Supply voltage tolerance is a DC rating, not a glitch rating.

For any clocked CMOS, and most especially for VLSI CMOS, V_{cc} decoupling is critical. CMOS draws current in extremely sharp spikes at the clock edges. The VHF and UHF components of these spikes are not drawn from the power supply, but from the decoupling capacitor. If the decoupling circuit is not sufficiently low in inductance, V_{cc} will glitch at each clock edge. A 0.1-μF decoupler cap should be used in a minimum-inductance configuration with the microcontroller. A minimum-inductance configuration minimizes the area of the loop formed by the chip (V_{cc} to V_{ss}), the traces to the decoupler cap, and the decoupler cap. PCB designers too often fail to understand that if the traces that connect the decoupler cap to the V_{cc} and V_{ss} pins are not short and direct, the decoupler loses much of its effectiveness.

Overshoot and ringing in signal lines are potential sources of logic upsets. These can largely be controlled by circuit layout. Inserting small resistors (about 100 ohms) in series with signal lines that seem to need them will also help.

The sharp edges produced by high-speed CMOS can cause RFI problems. The severity of these problems is largely a function of the PCB layout. All RFI problems are not necessarily solved by a better PCB layout. It may well be, for example, that in some RFI-sensitive designs,

high-speed CMOS is simply not the answer. But circuit layout is a critical factor in the noise performance of any electronic system, and more so in high-speed CMOS systems than others.

Circuit layout techniques for minimizing noise susceptibility and generation are discussed in References 3 and 4.

UNUSED PINS

CMOS input pins should not be left to float, but should always be pulled to one logic level or the other. If they float, they tend to float into the transition region between 0 and 1, where pull-up and pull-down devices in the input buffer are both conductive. This causes a significant increase in I_{cc}. A similar effect exists in NMOS circuits, but with less noticeable results.

In 80C51BH and 80C31BH designs, unused pins of Ports 1, 2, and 3 can be ignored, because they have internal pull-ups that will hold them at a valid logic-1 level. Port 0 pins are different, however; they do not have internal pull-ups (except during bus operations).

When the 80C51BH is in reset, the Port 0 pins are in a float state unless they are externally pulled up or down. If the device is to be held in reset for just a short time, the transient float state can probably be ignored. When the device comes out of reset, the pins stay afloat unless they are externally pulled either up or down. Alternatively, the software can internally write 0s to whatever Port 0 pins may be unused.

The same considerations are applicable to the 80C31BH when it is in reset. But when the 80C31BH comes out of reset, it commences bus operations, during which the logic levels at all pins are always well defined as high or low.

When the 80C31BH is in the Power-Down and Idle modes, however, it is not fetching instructions, and the Port 0 pins will float if not externally pulled high or low. The choice of whether to pull them high or low is the designer's. Normally it is sufficient to pull them up to V_{cc} with 10K resistors. But if power is going to be removed from circuits that are connected to the bus, it will be advisable to pull the bus pins down (normally with 10K resistors). Considerations involved in selecting pull-up and pull-down resistor values are as follows.

PULL-UP RESISTORS

If a pull-up resistor is to be used on a Port 0 pin, its minimum value is determined by I_{OL} requirements. If the pin is trying to emit a 0, then it will have to sink the current from the pull-up resistor plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 7-1a, while maintaining a valid output low (V_{OL}).
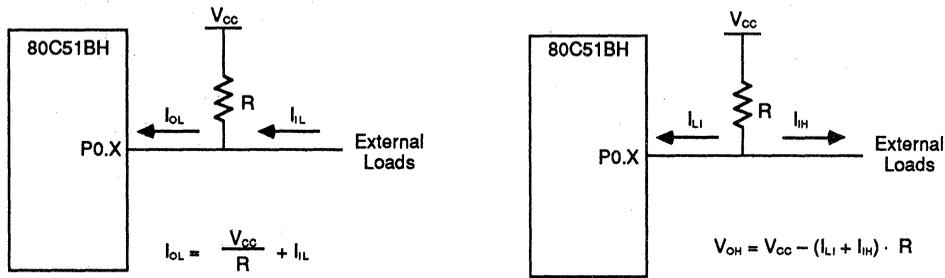
a. Minimum Value (P0.X is emitting a logic low)   b. Maximum Value (P0.X is in a high impedance state)

Figure 7-1. Conditions Defining Values of Pull-Up Resistor R
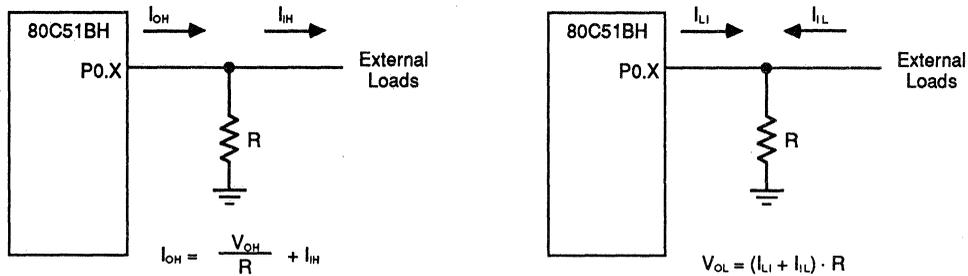
a. Minimum Value (P0.X is emitting a 1 in a   b. Maximum Value (P0.X is in a high impedance state)
   bus operation)

Figure 7-2. Conditions Defining Values of Pull-Down Resistor R

To guarantee that the pin voltage will not exceed 0.45 V, the resistor should be selected so that I_{OL} does not exceed the value specified on the data sheet. In most CMOS applications, the minimum value would be about 2K.

The maximum value would depend on how fast the pin must pull up after bus operations have ceased, and how high the V_{OH} level must be. The smaller the resistor, the faster it pulls up. Its effect on the V_{OH} level is that $V_{OH} = V_{CC} - (I_{LI} + I_{IH}) \cdot R$. I_{LI} is the input leakage current to the Port 0 pin, and I_{IH} is the input high current to the external loads, as shown in Figure 7-1b. Normally V_{OH} can be expected to reach 0.9 V_{CC} if the pull-up resistance does not exceed about 50K.

PULL-DOWN RESISTORS

If a pull-down resistor is to be used on a Port 0 pin, its minimum value is determined by V_{OH} requirements dur-

ing bus operations, and its maximum value is, in most cases, determined by leakage current.

During bus operations, the port uses internal pull-ups to emit 1s. The DC Characteristics in the data sheet list guaranteed V_{OH} levels for given I_{OH} currents. (The "–" sign in the I_{OH} value means the pin is sourcing that current to the external load, as shown in Figure 7-2.) To ensure the V_{OH} level listed in the data sheet, the resistor has to satisfy

$$\frac{V_{OH}}{R} + I_{IH} \leq |I_{OH}|$$

where I_{IH} is the input high current to the external loads.

When the pin goes into a high-impedance state, the pull-down resistor will have to sink leakage current from the pin, plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 7-2b. The

Port 0 leakage current is I_{LI} on the data sheet. The resistor should be selected so that the voltage developed across it by these currents will be seen as a logic low by whatever circuits are connected to it (including the 80C51BH). In CMOS/CMOS applications, 50K is normally a reasonable maximum value.

DRIVE CAPABILITY OF THE INTERNAL PULL-UPS

There is an important difference between NMOS and CMOS port drivers. The pins of Ports 1, 2, and 3 of the CMOS parts each have three pull-ups: strong, normal, and weak, as shown in Figure 7-3. The strong pull-up (P1) is only used during 0-to-1 transitions, to hasten the transition. The weak pull-up (P2) is on whenever the bit latch contains a 1. The "normal" pull-up (P3) is controlled by the pin voltage itself.

The reason that P3 is controlled by the pin voltage is that if the pin is being used as an input, and the external source pulls it to a low, then turning off P3 makes for a lower I_{IL}. The data sheet shows an "I_{TL}" specification. This is the current that P3 will source during the time the pin voltage is making its 1-to-0 transition. This is what I_{IL} would be if an input low at the pin did not turn P3 off.

Note, however, that this P3 turn-off mechanism puts a restriction on the drive capacity of the pin if it's being used as an output. If you're trying to output a logic high, and the external load pulls the pin voltage below the pin's V_{IH} min spec, P3 might turn off, leaving only the weak P2 to provide drive to the load. To prevent this from happening, you need to ensure that the load does not draw more than the I_{OH} spec for a valid V_{OH}. The idea is to make sure the pin voltage never falls below its own V_{IH} min specification.

POWER CONSUMPTION

The main reason for going to CMOS, of course, is to conserve power. There are other reasons, but this is the main one. Conserving power does not mean just reducing the electric bill; nor does it necessarily relate to battery operation, although battery operation without CMOS is pretty unhandy. The main reason for conserving power is to be able to put more functionality into a smaller space. Reduced power consumption allows the use of smaller and lighter power supplies. With less heat generated, denser packaging of circuit components is possible, and expensive fans and blowers can usually be eliminated. A cooler running chip is also more reliable, since most random and wearout failures relate to die
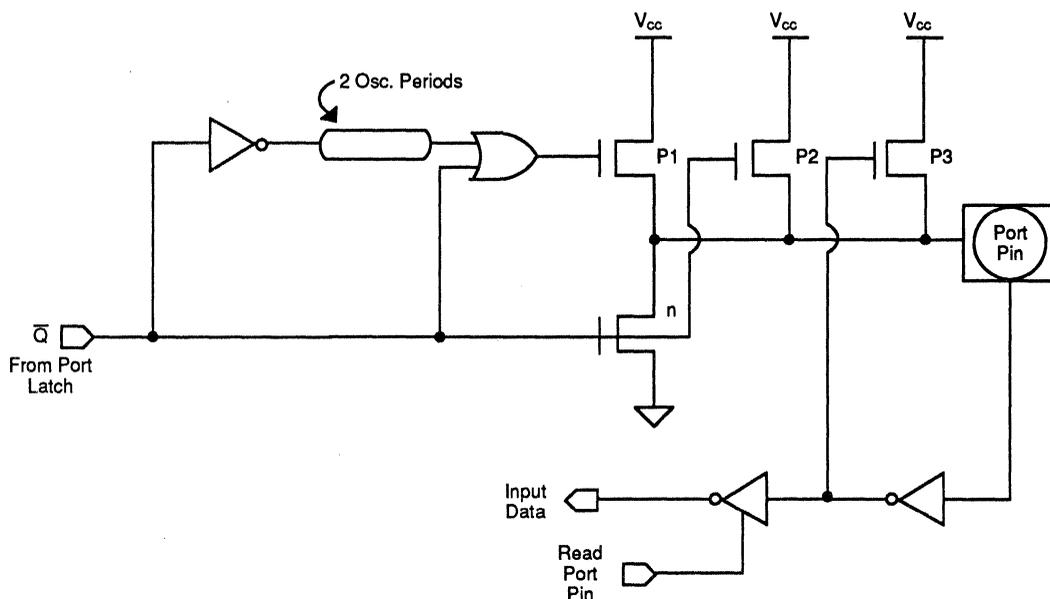

Figure 7-3. 80C51BH Output Drivers for Ports 1, 2, and 3

temperature. And finally, the lower power dissipation allows more functions to be integrated onto the chip.

CMOS consumes less power than NMOS because when CMOS is in a stable state, there is no path of conduction from Vcc to Vss except through various leakage paths. CMOS does draw current when it is changing states. How much current is drawn depends on how often and how quickly CMOS changes states.

During logical transitions, CMOS circuits draw current in sharp spikes that are made up of two components. One is the current that flows during the transition time when pull-up and pull-down FETs are both active. The average (DC) value of this component is larger when the transition times of the input signals are longer. For this reason, if the current draw is a critical factor in the design, slow rise and fall times should be avoided, even when the system speed does not seem to justify a need for nanosecond switching speeds.

The other component is the current that charges stray and load capacitance at the nodes of a CMOS logic gate. The average value of this current spike is its area (integral over time) multiplied by its repetition rate. Its area is the amount of charge it takes to raise the node capacitance, C, to Vcc. That amount of charge is just $C \cdot Vcc$. So the average value of the current spike is $C \cdot Vcc \cdot f$, where f is the clock frequency. This component of current increases linearly with clock frequency.

Keep in mind, though, that the other component of current is due to slow rise and fall times. A sinusoid is not the optimal waveform with which to drive the XTAL1 pin. Yet crystal oscillators, including the one on the 80C51BH, generate sinusoidal waveforms. Therefore, if the on-chip oscillator is being used, the device will draw more current at 500 kHz than it does at 1.5 MHz, as shown in Figure 7-4. If a good sharp square wave is derived from an external oscillator and is used to drive XTAL1, the microcontroller will draw less current. But the external oscillator will probably make up the difference.

The 80C51BH has two power-saving features not available in the NMOS devices: Idle and Power-Down modes of operation. The on-chip hardware that implements
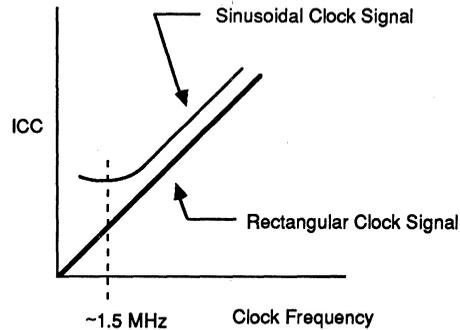
Figure 7-4. 80C51BH ICC versus Clock Frequency

these reduced power modes is shown in Figure 7-5. Both modes are invoked by software.

Idle: In the Idle Mode (IDL = 0 in Figure 7-5), the CPU puts itself to sleep by gating off its own clock. It does not stop the oscillator; it just stops the internal clock signal from getting to the CPU. Since the CPU draws 80 to 90 percent of the chip's power, shutting it off represents a fairly significant power savings. The on-chip peripherals (timers, serial port, interrupts, etc.) and RAM continue to function as normal. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

The Idle Mode is invoked by setting bit 0 (IDL) of the PCON register. PCON is not bit-addressable, so the bit has to be set by a byte operation, such as

    ORL   PCON,#1

The PCON register also contains flag bits GF0 and GF1, which can be used for any general purposes, or to give an indication if an interrupt occurred during normal operation or during Idle. In this application, the instruction that invokes Idle also sets one or both of the flag bits. Their status can then be checked in the interrupt routines.
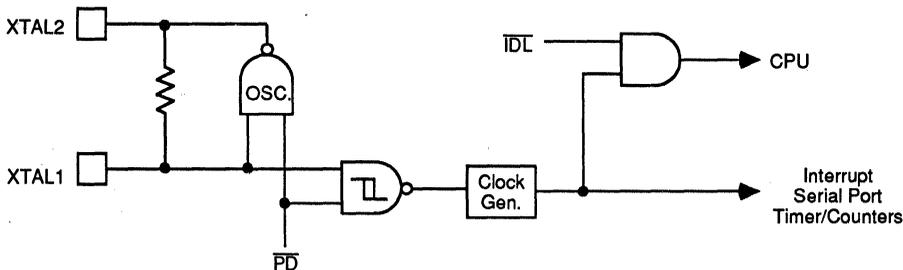
Figure 7-5. Oscillator and Clock Circuitry Showing Idle and Power-Down Hardware

While the device is in the Idle mode, ALE and $\overline{\text{PSEN}}$ emit logic high (V_{OH}), as shown in Table 7-3. This is so external EPROM can be deselected and have its output disabled.

The port pins hold the logical states they had at the time the Idle was activated. If the device was executing out of external program memory, Port 0 is left in a high impedance state and Port 2 continues to emit the high byte of the program counter (using the strong pull-ups to emit 1s). If the device was executing out of internal program memory, Ports 0 and 2 continue to emit whatever is in the P0 and P2 registers.

There are two ways to terminate Idle. Activation of any enabled interrupt will cause the hardware to clear bit 0 of the PCON register, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that invoked Idle.

The other way is with a hardware reset. Since the clock oscillator is still running, RST only needs to be held active for two machine cycles (24 oscillator periods) to complete the reset. Note that this exit from Idle writes 1s to all the ports, initializes all SFRs to their reset values, and restarts program execution from location 0.

Power Down: In the Power-Down mode ($\overline{\text{PD}}$ = 0 in Figure 7-5), the CPU puts the whole chip to sleep by turning off the oscillator. In case it was running from an external oscillator, it also gates off the path to the internal phase generators, so no internal clock is generated even if the external oscillator is still running. The on-chip RAM, however, saves its data, as long as V_{CC} is maintained. In this mode, the only I_{CC} that flows is leakage, which is normally in the micro-amp range.

The Power-Down mode is invoked by setting bit 1 in the PCON register, using a byte instruction such as

ORL PCON,#2

While the device is in Power Down, ALE and $\overline{\text{PSEN}}$ emit lows (V_{OL}), as shown in Table 7-3. ALE and $\overline{\text{PSEN}}$ are designed to emit lows so that power can be removed from the rest of the circuit, if desired, while the 80C51BH is in its Power-Down mode.

The port pins continue to emit whatever data was written to them. Note that Port 2 emits its P2 register data even if execution was from external program memory. Port 0 also emits its P0 register data, but if execution was from external program memory, the P0 register data is FF. The oscillator is stopped, and the part remains in this state as long as V_{CC} is held, and until it receives an external reset signal.

The only exit from Power Down is a hardware reset. Since the oscillator was stopped, RST must be held active long enough for the oscillator to re-start and stabilize. Then the reset function initializes all the Special Function Registers (ports, timers, etc.) to their reset values, and re-starts the program from location 0. Therefore, timer reloads, interrupt enables, baud rates, port status, etc. need to be re-established. Reset does not affect the content of the on-chip data RAM. If V_{CC} was held during Power Down, the RAM data is still good.

USING THE POWER-DOWN MODE

The software-invoked Power-Down feature offers a means of reducing the power consumption to a mere trickle in systems that are to remain dormant for some period of time, while retaining important data. The user should give some thought to what state the port pins should be left in during the time the clock is stopped, and write those values to the port latches before invoking Power Down.

Table 7-3. Status of Pins in Idle and Power-Down Modes
("SFR data" means the port pins emit their internal register data;
"PCH" is the high byte of the program counter)

Pin	Internal Execution		External Execution	
	Idle	Power Down	Idle	Power Down
ALE	1	0	1	0
$\overline{\text{PSEN}}$	1	0	1	0
P0	SFR data	SFR data	high-Z	high-Z
P1	SFR data	SFR data	SFR data	SFR data
P2	SFR data	SFR daa	PCH	SFR data
P3	SFR data	SFR data	SFR data	SFR data

If Vcc is going to be held to the entire circuit, values should be written to the port latches that would deselect peripherals before invoking Power Down. For example, if external memory is being used, the P2 SFR should be loaded with a value that will not generate an active chip select to any memory device.

In some applications, Vcc to part of the system may be shut off during Power Down, so that even quiescent and standby currents are eliminated. Signal lines that connect to those chips must be brought to a logic low, whether the chip in question is CMOS, NMOS, or TTL, before Vcc is shut off to them. CMOS pins have parasitic pn junctions to Vcc, which will be forward biased if Vcc is reduced to zero while the pin is held at a logic high. NMOS pins often have FETs that look like diodes to Vcc. TTL circuits may actually be damaged by an input high if Vcc = 0. That is why the 80C51BH outputs low at ALE and PSEN during Power Down.

Figure 7-6 shows a circuit that can be used to turn Vcc off to part of the system during Power Down. The circuit will ensure that the secondary circuit is not de-energized until after the 80C31BH is in Power Down, and that the 80C31BH does not receive a reset (terminating the Power-Down mode) before the secondary circuit is re-energized. Therefore, the program memory itself can be part of the secondary circuit.

In Figure 7-6, when Vcc is switched on to the 80C31BH, capacitor C1 provides a power-on reset. The reset function writes 1s to all the port pins. The 1 at P2.6 turns Q1 on, enabling Vcc to the secondary circuit through transistor Q2. As the 80C31BH comes out of reset, Port 2 commences emitting the high byte of the Program Counter, which results in the P2.7 and P2.6 pins outputting 0s. The 0 at P2.7 ensures continuation of Vcc to the secondary circuit.

The system software must now write a 1 to P2.7 and a 0 to P2.6 in the Port 2 SFR, P2. These values will not appear at the Port 2 pins as long as the device is fetching instructions from external program memory. However, whenever the 80C31BH goes into Power Down, these values will appear at the port pins, and will shut off both transistors, disabling Vcc to the secondary circuit.
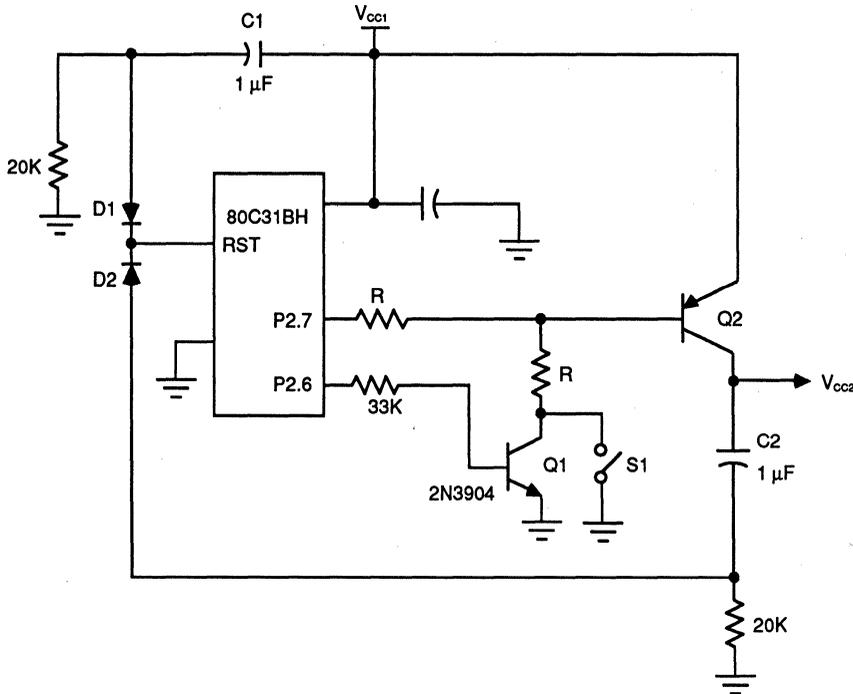
Figure 7-6. The 80C31BH De-energizes Part of the Circuit (Vcc2) During Power Down
(selections of R and Q2 depend on Vcc2 current draw)

Closing the switch S1 re-energizes the secondary circuit, and at the same time sends a reset through C2 to the 80C31BH to wake it up. The diode D1 is to prevent C1 from hogging current from C2 during this secondary reset. D2 prevents C2 from discharging through the RST pin when Vcc to the secondary circuit goes to zero.

USING POWER MOSFETS TO CONTROL Vcc

Power MOSFETs are gaining in popularity and availability. The easiest way to control Vcc is with a Logic Level pFET, as shown in Figure 7-7a. This circuit allows the full Vcc to be used to turn the device on. Unfortunately, power pFETs are not economically competitive with bipolar transistors of comparable ratings.

Power nFETs are both economical and available, and can be used in this application if a DC supply of higher voltage is available to drive the gate. Figure 7-7b shows how to implement a Vcc switch using a power nFET and a (nominally) +12-V supply. The problem here is that if the device is on, its source voltage is +5 V. To maintain the on state, the gate has to be another 5 or 10 V above that. The "12-V" supply is not particularly critical. A minimally filtered, unregulated rectifier will suffice.

BATTERY BACKUP SYSTEMS

Here we consider circuits that normally draw power from the AC line, but switch to battery operation in the event of a power failure. We assume that in battery operation high-current loads will be allowed to die along with the AC power. The system may continue then with reduced functionality, monitoring a control transducer, perhaps, or driving an LCD. Or it may go into a bare-bones survival mode, in which critical data is saved but nothing else happens until AC power is restored.

In any case, it is necessary to have some early warning of an impending power failure so that the system can arrange an orderly transfer to battery power. Early warning systems can operate by monitoring either the AC line voltage or the unregulated rectifier output, or even by monitoring the regulated DC voltage.

Monitoring the AC line voltage gives the earliest warning. That way you can know within one or two half-cycles of line frequency that AC power is down. In most cases you then have at least another half-cycle of line frequency before the regulated Vcc starts to fall. In a half-cycle of line frequency, an 80C51BH can execute about 5,000 instructions—plenty of time to arrange an orderly transfer of power.

The circuit in Figure 7-8 uses a Zener diode to test the line voltage each half cycle, and a junction transistor to pass the information on the 80C51BH. Obviously a voltage comparator with a suitable reference source can perform the same function, if one prefers. If the line voltage reaches an acceptably high level, it breaks over Z1, drives Q1 to saturation, and interrupts the 80C51BH. The interrupt would be transition-activated in this application. The interrupt service routine reloads one of the 80C51BH's timers to a value that will make it roll over in something between one and two half-cycles of line frequency. As long as the line voltage is healthy, the timer never rolls over, because it is reloaded every half cycle. If there is a single half cycle in which the line voltage does not reach a high enough level to generate the interrupt, the timer rolls over and generates a **timer** interrupt.

The timer interrupt then commences the transition to battery backup. Critical data needs to be copied into protected RAM. Signals to circuits that are going to lose power must be written to logic low. Protected circuits

a. Using a pFET   b. Using an nFET
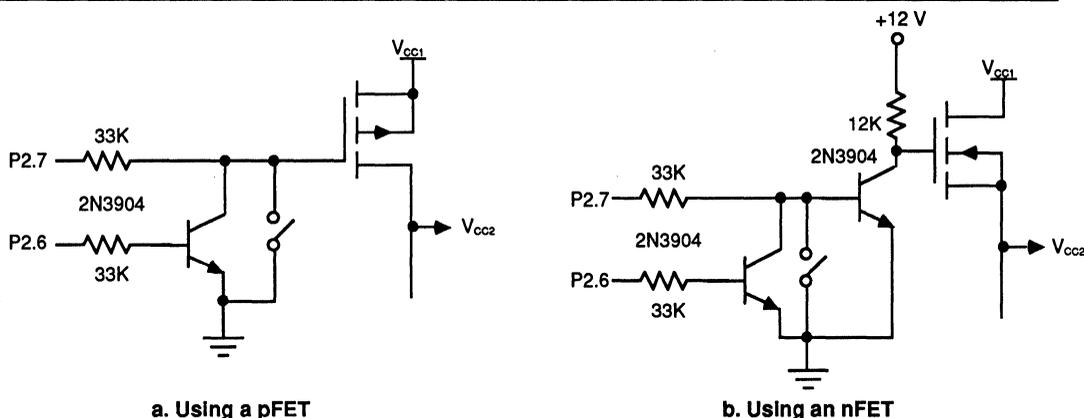
Figure 7-7. Using Power MOSFETs to Control Vcc2
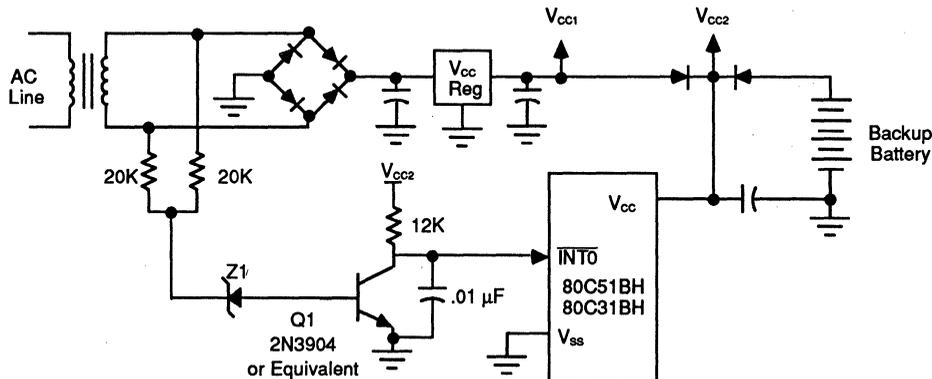
**Figure 7-8. Power Failure Detector with Battery Backup
(when AC power fails, Vcc1 goes down and Vcc2 is held)**

(those powered by V_{CC2}) that communicate with unprotected circuits must be deselected. The microcontroller itself may be put into Idle, so that it can continue some level of interrupt-driven functionality, or it may be put into Power Down.

Note that if the CPU is going to invoke Power Down, the Special Function Registers may also need to be copied into protected RAM, since the reset that terminates the Power-Down mode will also initialize all the SFRs to their reset values.

The circuit in Figure 7-8 does not show a wake-up mechanism. A number of choices are available, however. A pushbutton could be used to generate an interrupt, if the CPU is in Idle, or to activate reset, if the CPU is in Power Down.

Automatic wake-up on power restoration is also possible. If the CPU is in Idle, it can continue to respond to any interrupts that might be generated by Q1. The interrupt service routine determines from the status of flag bits GF0 and GF1 in PCON that it is in Idle because there was a power outage. It can then sample V_{CC1} through a voltage comparator similar to Z1, Q1 in Figure 7-8. A satisfactory level of V_{CC1} would be indicated by the transistor being in saturation.

But perhaps the timer, that is the key to the operation of the circuit in Figure 7-8, cannot be spared. In that case a retriggerable one-shot, triggered by the AC line voltage, can perform essentially the same function. Figure 7-9 shows an example of this type of power-failure detector. A retriggerable one-shot (one half of a 74HC123) monitors the AC line voltage through transistor Q1. Q1 retriggers the one-shot every half cycle of line frequency. If the output pulse width is between one and two half-cycles of line frequency, then a single missing or low half cycle will generate an active low warning flag, which can be used to interrupt the microcontroller.

The interrupt routine takes care of the transition to battery back-up. From this point, V_{CC1} may or may not actually drop out. The missing half-cycle of line voltage that caused the Power-Down sequence may have been nothing more than a short glitch. If the AC line comes back strong enough to trigger the one-shot while V_{CC1} is still up (as indicated by the state of transistor Q2), then the other half of the 74HC123 will generate a wake-up signal.

Having been awakened, the 80C51BH will stay awake for at least another half-cycle of line frequency (another 5,000 or so instructions) before possibly being told to arrange another transfer of power. Consequently, if the line voltage is jittering erratically around the switchover point (determined by diode Z1), the system will limp along executing in half-cycle units of line frequency.

On the other hand, if the power outage is real and lengthy, V_{CC1} will eventually fall below the level at which the backup battery takes over. The backup battery maintains power to the 80C51BH, the 74HC123, and to whatever other circuits are being protected during this outage. The battery voltage must be high enough to maintain $V_{CC(min)}$ specs to the 80C51BH.

If the microcontroller is an 80C31BH, executing out of external ROM, and if the 80C31BH is put into Idle during the power outage, then the external ROM must also be supplied by the battery. On the other hand, if the 80C31BH is put into Power Down during the outage, then the ROM can be allowed to die with the AC power.
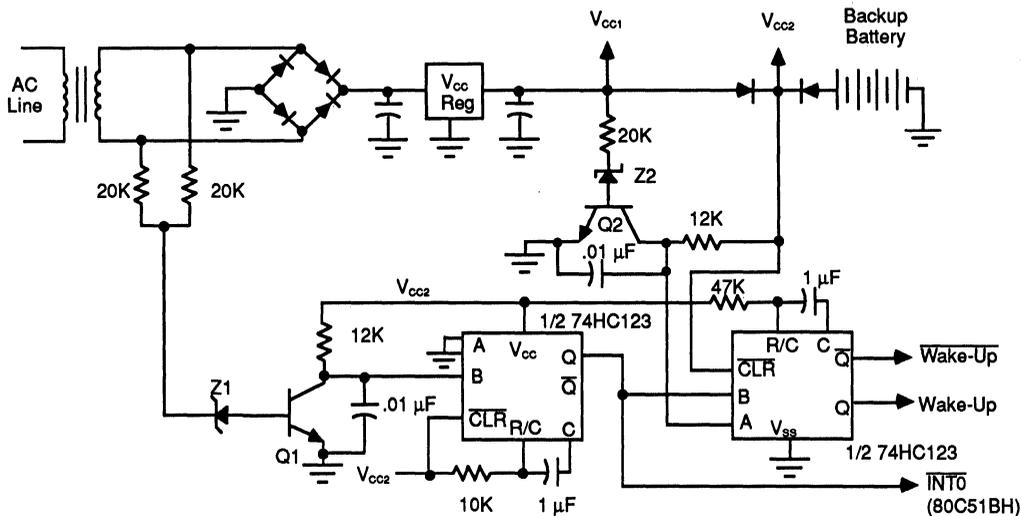
Figure 7-9. Power Failure Detector uses retriggerable one-shots to flag impending power outage and generate automatic wake-up when power returns

The considerations here are the same as in Figure 7-6: Vcc to the ROM is still up at the time Power Down is invoked, and we must ensure (through selection of diode Z2 in Figure 7-9) that the 80C31BH is not awakened until ROM power is back in spec.

POWER SWITCHOVER CIRCUITS

Battery backup systems need to have a way for the protected circuits to draw power from the line-operated power supply when that source is available, and to switch over to battery power when required. The switchover circuit is simple if the entire system is to be battery pow-

ered in the event of a line power outage. In that case a pair of diodes suffice, as shown in Figure 7-9, provided $Vcc_{(min)}$ specs are still met after the diode drop has been subtracted from its respective power source.

The situation becomes more complicated when part of the circuit is going to be allowed to die when the AC power goes out. In that case it is difficult to maintain equal Vccs to protected and unprotected circuits (and possibly dangerous not to). The problem can be alleviated by using a Schottky diode instead of a 1N4001, for its lower forward voltage drop. The 1N5820, for example,

a. Using a PNP Transistor

b. Using a Power MOSFET
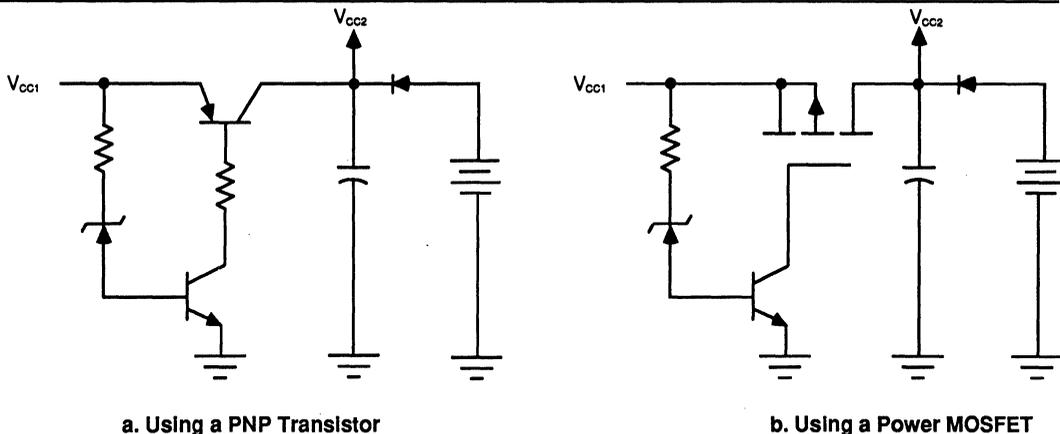
Figure 7-10. Power Switchover Circuits

has a forward drop of about 0.35 V at 1A. Other solutions are to use a transistor or power MOSFET switch, as shown in Figure 7-10. With minor modifications this switch can be controlled by port pins.

80C31BH + CMOS EPROM

The 27C256 is AMD's 32K-byte CMOS EPROM. It requires an external address latch, and can be used with the 80C31BH as shown in Figure 7-11a. In most 8031 + 27256 (NMOS) applications, the Chip Enable ($\overline{\text{CE}}$) pin is hardwired to ground (since it's normally the only program memory on the bus). This can be done with the CMOS versions as well, but there is some advantage in connecting $\overline{\text{CE}}$ to ALE, as shown in Figure 7-11. The advantage is that if the 80C31BH is put into Idle mode, since ALE goes to a 1 in that mode, the 27C256 will be deselected and go into a low-current standby mode.

The timing waveforms for this configuration are shown in Figure 7-11b. The signals and timing parameters in parentheses are those of the 27C256 and the others are of the 80C31HB, except Tprop is a parameter of the address latch. The requirements for timing compatibility are

TAVIV – Tprop > tACC

TLLIV > t$\overline{\text{CE}}$

TPLIV > tOE

TPXIZ > tDF

If the application is going to use the Power-Down mode there is another consideration; in Idle, ALE = $\overline{\text{PSEN}}$ = 1,

and in Power Down, ALE = $\overline{\text{PSEN}}$ = 0. In a realistic application there are likely to be more chips in the circuit than are shown in Figure 7-11, and it is likely that the nonessential ones will have their Vcc removed while the CPU is in Power Down. In that case the EPROM and the address latch should be among the chips that have Vcc removed, and logic lows are exactly what are required at ALE and $\overline{\text{PSEN}}$.

But if Vcc is going to be maintained to the EPROM during Power Down, then it will be necessary to deselect the EPROM when the CPU is in Power Down. If Idle is never invoked, $\overline{\text{CE}}$ of the EPROM can be connected to P2.7 of the 80C31BH, as shown in Figure 7-12a. In normal operation, P2.7 will be emitting the MSB of the Program Counter, which is 0 if the program contains less than 32K of code. Then when the CPU goes into Power Down, the Port 2 pins emit P2 SFR data, which puts a 1 at P2.7, thus deselecting the EPROM.

If Idle and Power Down are both going to be used, $\overline{\text{CE}}$ of the EPROM can be driven by the logical OR of ALE and P2.7, as shown in Figure 7-12b. In Idle, ALE = 1 will deselect the EPROM, and in Power Down, P2.7 = 1 will deselect it.

Pull-down resistors are shown in Figure 7-11 under the assumption that something on the bus is going to have its Vcc removed during Power Down. If this is not the case, pull-ups can be used as well as pull-downs.
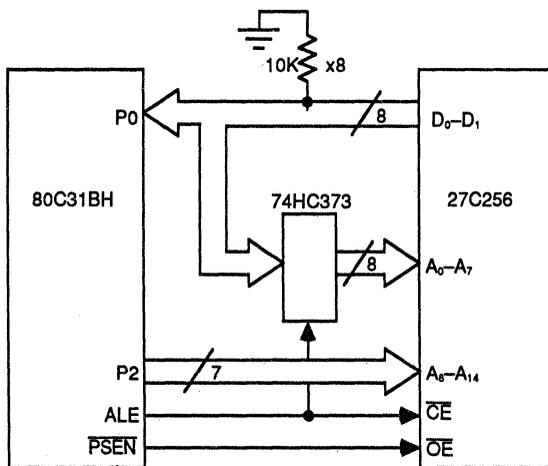
a. Circuit
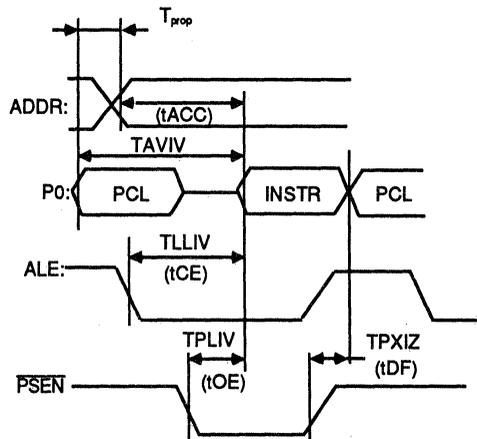
b. Timing Waveforms

Figure 7-11. 80C31BH + 27C256

a. Power Down Is Used but Not Idle   b. Idle and Power Down Both Used

Figure 7-12. Modifications to 80C31/27C256 Interface

SCANNING A KEYBOARD

There are many different kinds of keyboards, but alpha-numeric keyboards generally consist of a matrix of eight scan lines and eight receive lines as shown in Figure 7-13. Each set of lines connects to one port of the microcontroller. The software has written 0s to the scan lines, and 1s to the receive lines. Pressing a key connects a scan line to a receive line, thus pulling the receive line to a logic low.

The eight receive lines are ANDed to one of the external interrupt pins, so that pulling any of the receive lines low generates an interrupt. The interrupt service routine has to identify the pressed key, if only one key is down, and convert that information to some useful output. If more than one key in the line matrix is found to be pressed, no action is taken. (This is a "two key lock-out" scheme.)
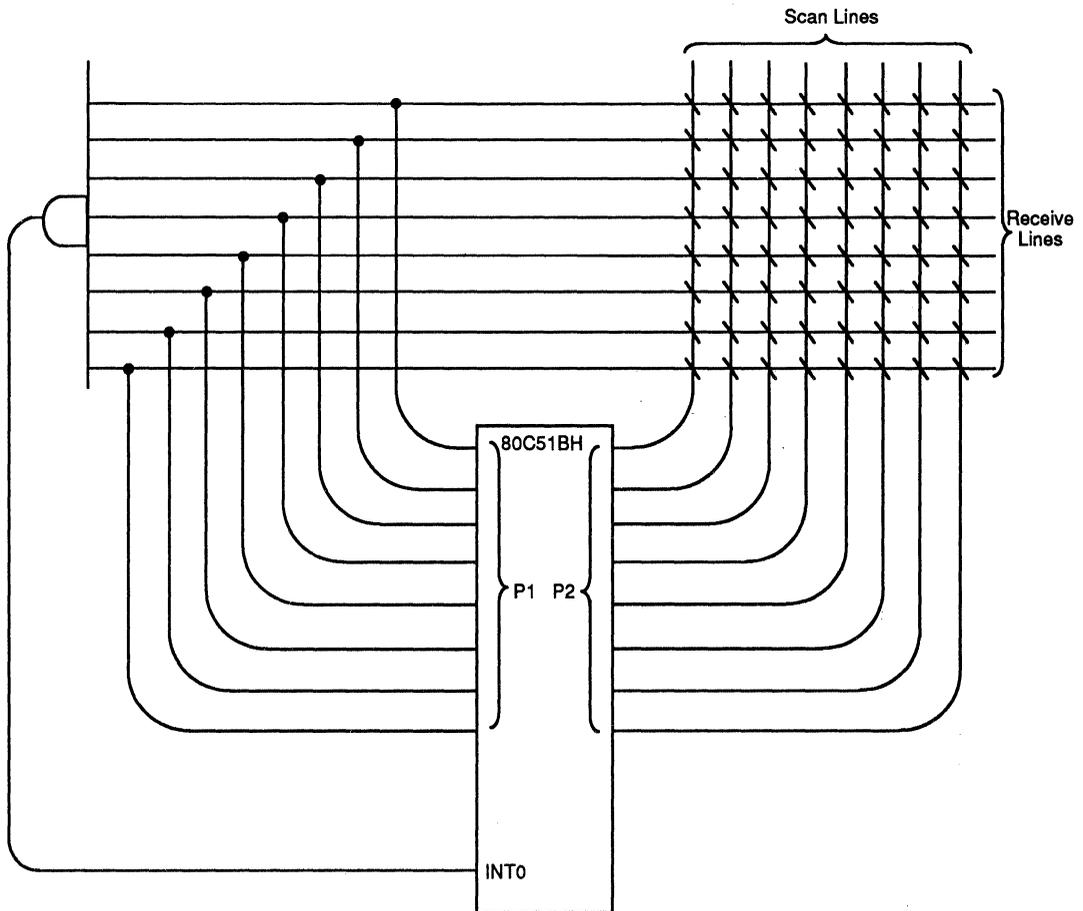
Figure 7-13. Scanning a Keyboard

On some keyboards, certain keys (Shift, Control, Escape, etc.) are not a part of the line matrix. These keys would connect directly to a port pin on the microcontroller, and would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly.

Normally the microcontroller would be in Idle mode when a key has not been pressed, and another task is not in progress. Pressing a matrix key generates an interrupt, which terminates the Idle. The interrupt service routine would first call a 30-ms (or so) delay to debounce the key, and then set about the task of identifying which key is down.

First, the current state of the receive lines is latched into an internal register. Then 0s are written to the receive lines and 1s to the scan lines, and the scan lines are read. If a single key is down, all but one of these lines would be read as 1s. By locating the single 0 in each set of lines, the pressed key can be identified. If more than one matrix key is down, one or both sets of lines will contain multiple 0s.

A subroutine is used to determine which of 8 bits in either set of lines is 0, and whether more than one bit is 0. Table 7-4 shows a subroutine (SCAN) that does that using the 8051 bit-addressing capability. To use the subroutine, move the line data into a bit-addressable RAM location named LINE, and call the SCAN routine. The number of LINE bits that are zero is returned in ZERO_COUNTER. If only one bit is zero, its number (1 through 8) is returned in ZERO_BIT.

The interrupt service routine that is executed in response to a key closure might then be as follows:

```
RESPONSE_TO_KEY_CLOSURE:
        CALL    DEBOUNCE_DELAY
        MOV     LINE,P1;        ;See Figure 9-16.
        CALL    SCAN
        DJNZ    ZERO_COUNTER,REJECT
        MOV     ADDRESS,ZERO_BIT
        MOV     P2,#0FFH;       ;See Figure 9-16.
        MOV     P1,#0
        MOV     LINE,P2
        CALL    SCAN
        DJNZ    ZERO_COUNTER,REJECT
        XCH     A,ZERO_BIT
        SWAP    A
        ORL     ADDRESS,A
        XCH     A,ZERO_BIT
        MOV     P1,#0FFH
        MOV     P2,#0
REJECT: CLR     EX0
        RETI
```

Notice that RESPONSE_TO_KEY_CLOSURE does not change the Accumulator, the PSW, nor any of the registers R0 through R7. Neither do SCAN or

Table 7-4. Subroutine SCAN Determines which of Eight Bits in LINE is 0

```
SCAN:   MOV     ZERO_COUNTER,#0    ;   ZERO_COUNTER counts the number of 0s in LINE.
        JB      LINE.0,ONE         ;   Test LINE bit 0.
        INC     ZERO_COUNTER       ;   If LINE.0 = 0, increment ZERO_COUNTER
        MOV     ZERO_BIT,#1        ;     and record that line number 1 is active.
ONE:    JB      LINE.1,TWO         ;   Procedure continues for other LINE bits.
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#2        ;   Line number 2 is active.
TWO:    JB      LINE.2,THREE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#3        ;   Line number 3 is active.
THREE:  JB      LINE.3,FOUR
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#4        ;   Line number 4 is active.
FOUR:   JB      LINE.4,FIVE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#5        ;   Line number 5 is active.
FIVE:   JB      LINE.5,SIX
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#6        ;   Line number 6 is active.
SIX:    JB      LINE.6,SEVEN
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#7        ;   Line number 7 is active.
SEVEN:  JB      LINE.7,EIGHT
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#8        ;   Line number 8 is active.
EIGHT:  RET
```

DEBOUNCE_DELAY. The result is a one-byte key address (ADDRESS) that identifies the pressed key. The key's scan line number is in the upper nibble of ADDRESS, and its receive line number is in the lower nibble. ADDRESS can be used in a look-up table to generate a key code to transmit to a host computer, and/or to a display device.

The keyboard interrupt itself must be edge-triggered, rather than level-activated, so that the interrupt routine is invoked when a key is pressed, and is not constantly being repeated as long as the key is held down. In edge-triggered mode, the on-chip hardware clears the interrupt flag (EX0, in this case) as the service routine is being vectored to. In this application, however, contact bounce will cause several more edges to occur after the service routine has been vectored to, during the DEBOUNCE_DELAY routine. Consequently, it is necessary to clear EX0 again in software before executing RETI.

The debounce delay routine also takes advantage of the Idle mode. In this routine a timer must be preloaded with a value appropriate to the desired length of delay. This value would be

$$\text{timer preload} = -\frac{(\text{OSC kHz}) \cdot (\text{delay time } \mu s)}{12}$$

For example, with a 3.58-MHz oscillator frequency, a 30-ms delay could be obtained using a preload value of −8950, or DD0A, in hex digits.

In the debounce delay routine (Table 7-5), the timer interrupt is enabled and set to a higher priority than the keyboard interrupt, because as we invoke Idle, the keyboard interrupt is still "in progress." An interrupt of the same priority will not be acknowledged, and will not terminate the Idle mode. With the timer interrupt set to priority 1 while the keyboard interrupt is a priority 0, the timer interrupt, when it occurs, will be acknowledged and will wake up the CPU. The timer interrupt service routine

does not itself have to do anything. The service routine might be nothing more than a single RETI instruction. RETI from the timer interrupt service routine then returns execution to the debounce delay routine, which shuts down the timer and returns execution to the keyboard service routine.

DRIVING AN LCD

An LCD (Liquid Crystal Display) consists of a backplane and any number of segments or dots that will be used to form the displayed image. Applying a voltage (nominally 4 to 5 V) between any segment and the backplane causes the segment to darken. The only catch is that the polarity of the applied voltage has to be periodically reversed, or else a chemical reaction takes place in the LCD that causes deterioration and eventual failure of the liquid crystal.

To prevent this from happening, the backplane and all the segments are driven with an AC signal, which is derived from a rectangular voltage waveform. If a segment is to be "off," it is driven by the same waveform as the backplane. Thus, it is always at backplane potential. If the segment is to be "on," it is driven with a waveform that is the inverse of the backplane waveform. Thus, it has about 5 V of periodically changing polarity between it and the backplane.

With a little software overhead, the 80C51BH can perform this task without the need for additional LCD drivers. The only drawback is that each LCD segment uses up one port pin, and the backplane uses one more. If more than, say, two 7-segment digits are being driven, there are not many port pins left for other tasks. Nevertheless, assuming a given application leaves enough port pins available to support this task, the considerations for driving the LCD are as follows.

Suppose, for example, it is a 2-digit display with a decimal point. One port (TENS_DIGIT) connects to the seven segments of the tens digit plus the backplane.

Table 7-5. Subroutine DEBOUNCE_DELAY Puts the 80C51BH into Idle During the Delay Time

```
DEBOUNCE_DELAY:
        MOV     TL1,#TL1_PRELOAD    ;   Preload low byte.
        MOV     TH1,#TH1_PRELOAD    ;   Preload high byte.
        SETB    ET1                 ;   Enable Timer 1 interrupt.
        SETB    PT1                 ;   Set Timer 1 interrupt to high priority.
        SETB    TR1                 ;   Start timer running.
        ORL     PCON,#1             ;   Invoke Idle mode.
;
;   The next instruction will not be executed until the delay times out.
;
        CLR     TR1                 ;   Stop the timer.
        CLR     PT1                 ;   Back to priority 0 (if desired).
        CLR     ET1                 ;   Disable Timer 1 interrupt (if desired).
        RET                         ;   Continue keyboard scan.
```

Another port (ONES_DIGIT) connects to a decimal point plus the seven segments of the ones digit.

One of the 80C51BH timers is used to mark off half-periods of the drive voltage waveform. The LCD drive waveform should have a rep rate between 30 and 100 Hz, but it's not very critical. A half-period of 12 ms will set the rep rate to about 42 Hz. The preload/reload value to get 12 ms to rollover is the 2's complement negative of the oscillator frequency in kHz: If the oscillator frequency is 3.58 MHz, the reload value is –3850, or F204 in hex digits.

Now, the 80C51BH would normally be in Idle, to conserve power, during the time that the LCD and other tasks are not requiring servicing. When the timer rolls over, it generates an interrupt that brings the 80C51BH out of Idle. The service routine reloads the timer (for the next rollover), and inverts the logic levels of all the pins that are connected to the LCD. It might look like this:

```
LCD_DRIVE_INTERRUPT:
    MOV    TL1,#LOW(-XTAL_FREQ)
    MOV    TH1,#HIGH(-XTAL_FREQ)
    XRL    TENS_DIGIT,#0FFH
    XRL    ONES_DIGIT,#0FFH
    RETI
```

To update the display, one would use a look-up table to generate the characters. In the table, "on" segments are represented as 1s, and "off" segments as 0s. The backplane bit is represented as a 0. The quantity to be displayed is stored in RAM as a BCD value. The look-up table operates on the low nibble of the BCD value, and produces the bit pattern that is to be written to either the ones digit or the tens digit. Before the new patterns can be written to the LCD, the LCD drive interrupt has to be disabled. That is to prevent a polarity reversal from taking place between the times the two digits are written. The update subroutine is shown in Table 7-6.

USING AN LCD DRIVER

As was noted, driving an LCD directly with an 80C51BH uses a lot of port pins. LCD drivers are available in CMOS to interface an 80C51BH to a 4-digit display using only seven of the 80C51BH's I/O pins. Basically, the 80C51BH tells the LCD driver what digit is to be displayed (four bits) and what position it is to be displayed in (two bits), and toggles a Chip Select pin to tell the driver to latch this information. The LCD driver generates the display characters (hex digits), and takes care of the polarity reversals using its own RC oscillator to generate the timing. Figure 7-14 shows an 80C51BH working with an ICM7211M to drive a 4-digit LCD; the software that updates the display is shown in Table 7-7.

One could equally well send information to the LCD driver over the bus by setting up the Accumulator with the digit select and data input bits, and executing a MOVX @ R0,A instruction. The LCD-driver chip select would be driven by the CPU $\overline{WR}$ signal. This is a little easier in software than the direct bit manipulation shown in Figure 7-14. However, it uses more I/O pins, unless there is already some external memory involved. In that case, no extra pins are used up by adding the LCD driver to the bus.

RESONANT TRANSDUCERS

Analog transducers are often used to convert the value of a physical property, such as temperature, pressure, etc., to an analog voltage. These kinds of transducers then require an analog-to-digital converter to put the measurement into a form that is compatible with a digital control system. Another kind of transducer is now becoming available that encodes the value of the physical property into a signal that can be directly read by a digital control system. These devices are called resonant transducers.

Table 7-6. UPDATE_LCD Routine Writes Two Digits to an LCD

```
UPDATE_LCD:
    CLR    ET1                    ;  Disable LCD drive interrupt.
    MOV    DPTR,#TABLE_ADDRESS    ;  Look-up table begins at TABLE_ADDRESS.
    MOV    A,BCD_VALUE            ;  Digits to be displayed.
    SWAP   A                      ;  Move tens digit to low nibble.
    ANL    A,#0FH                 ;  Mask off high nibble.
    MOVC   A,@A+DPTR              ;  Tens digit pattern to accumulator.
    MOV    TENS_DIGIT,A           ;  Update LCD tens digit.
    MOV    A,BCD_VALUE            ;  Digits to be displayed.
    ANL    A,#0FH                 ;  Mask off tens digit.
    MOVC   A,@A+DPTR              ;  Ones digit pattern to accumulator.
    MOV    C,DECIMAL_POINT        ;  Add decimal point to segment
    MOV    ACC.7,C                ;      pattern. Update LCD decimal point
    MOV    ONES_DIGIT,A           ;      and ones digit.
    SETB   ET1                    ;  Re-enable LCD drive interrupt.
    RET
```
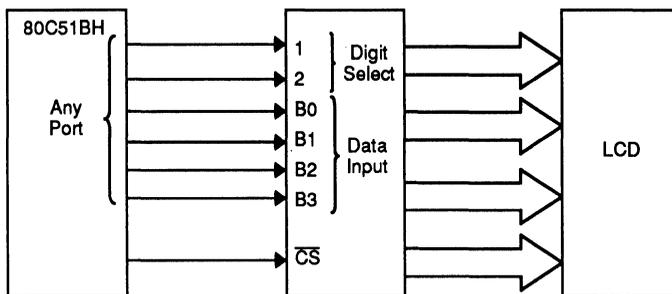
Figure 7-14. Using an LCD Driver

Resonant transducers are oscillators whose frequency depends in a known way on the physical property being measured. These devices output a train of rectangular pulses whose repetition rate encodes the value of the quantity being measured. The pulses can in most cases be fed directly into the 80C51BH, which then measures either the frequency or period of the incoming signal, basing the measurement on the accuracy of its own clock oscillator. The 80C51BH can even do this in its sleep, that is, in Idle.

When the frequency or period measurement is completed, the 80C51BH wakes itself up for a very short time to perform a sanity check on the measurement and convert it in software to any scaling of the measured quantity that may be desired. The software conversion can include corrections for nonlinearities in the transducer's transfer function.

Table 7-7. UPDATE_LCD Routine Writes Four Digits to an LCD Driver

```
UPDATE_LCD:
        MOV       A,DISPLAY_HI        ;  High byte of 4-digit display.
        SETB      DIGIT_SELECT_2      ;  Select leftmost digit of LCD.
        SETB      DIGIT_SELECT_1      ;     (digit address = 11B.)
        CALL      SHIFT_AND_LOAD      ;  High nibble of high byte to selected digit.
        CLR       DIGIT_SELECT_1      ;  Select second digit of LCD (address = 10B).
        CALL      SHIFT_AND_LOAD      ;  Low nibble of high byte to selected digit.
        MOV       A,DISPLAY_LO        ;  Low byte of 4-digit display.
        CLR       DIGIT_SELECT_2      ;  Select third digit of LCD.
        SETB      DIGIT_SELECT_1      ;     (digit address = 01B.)
        CALL      SHIFT_AND_LOAD      ;  High nibble of low byte to selected digit.
        CLR       DIGIT_SELECT_1      ;  Select fourth digit (address = 00B).
        CALL      SHIFT_AND_LOAD      ;  Low nibble of low byte to selected digit.
        RET
SHIFT_AND_LOAD
        RLC       A                   ;  MSB to carry bit (CY).
        MOV       DAT_INPUT_B3,C      ;  CY to Data Input pin B3.
        RLC       A                   ;  Next bit to CY.
        MOV       DATA_INPUT_B2,C     ;  CY to Data Input pin B2.
        RLC       A                   ;  Next bit to CY.
        MOV       DATA_INPUT_B1,C     ;  CY to Data Input pin B1.
        RLC       A                   ;  Last bit to CY.
        MOV       DATA_INPUT_B0,C     ;  CY to Data Input pin B0.
        CLR       CHIP_SELECT         ;  Toggle Chip Select.
        SETB      CHIP_SELECT         ;  0-to-1 transition latches info.
        RET
```

Resolution is also controlled by software, and can even be dynamically varied to meet changing needs as a situation becomes more critical. For example, in a process controller, resolution can be increased ("fine tune" the control) as the process approaches its target.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz to 500 kHz, depending on the design. Transducers are available that have a full-scale frequency shift of 2 to 1. The transducer operates from a supply voltage range of 3 V to 20 V, which means it can operate from the same supply voltage as the 80C51BH. At 5 V, the transducer draws less than 5 mA (Reference 5). It can normally be connected directly to one of the 80C51BH port pins, as shown in Figure 7-15.
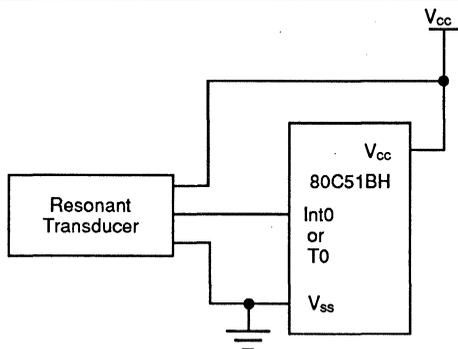
Figure 7-15. Resonant Transducer Does Not Require An A/D Converter

FREQUENCY MEASUREMENTS

Measuring a frequency means counting pulses for a known sample time. Two timer/counters can be used, one to mark off the sample time and one to count pulses. If the frequency being counted does not exceed 50 kHz or so, one may equally well connect the transducer signal to one of the external interrupt pins and count pulses in software. That frees up one timer, with very little cost in CPU time.

The count that is directly obtained is $T \cdot F$, where T is the sample time and F is the frequency. The full scale range is $T \cdot (Fmax - Fmin)$. For n-bit resolution

$$1 \text{ LSB} = \frac{T \cdot (Fmax - Fmin)}{2^n}$$

Therefore, the sample time required for n-bit resolution is

$$T = \frac{2^n}{Fmax - Fmin}$$

For example, 8-bit resolution in the measurement of a frequency that varies between 7 kHz and 9 kHz would require, according to this formula, a sample time of 128 ms. The maximum acceptable frequency count would be 128 ms · 9 kHz = 1152 counts. The minimum would be 896 counts. Subtracting 896 from each frequency count (or presetting the frequency counter to −896 = 0FC80H) would allow the frequency to be reported on a scale of 0 to FF in hex digits.

To implement the measurement, one timer is used to establish the sample time. The timer is preset to a value that causes it to roll over at the end of the sample time, generating an interrupt and waking the CPU from its Idle mode. The required preset value is the 2's complement negative of the sample time measured in machine cycles. The conversion from sample time to machine cycles is to multiply it by 1/12 the clock frequency. For example, if the clock frequency is 12 MHz, then a sample time of 128 ms is

(128 ms) · (12000 kHz)/12 = 128000 machine cycles.

Then the required preset value to cause the timer to roll over in 128 ms is

−128000 = FE0C00, in hex digits.

Note that the preset value is three bytes wide, whereas the timer is only two bytes wide. This means the timer must be augmented in software in the timer interrupt routine to three bytes. The 80C51BH has a DJNZ instruction (decrement and jump if not zero) that makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the 2's complement of what it would be for an up-counter. For example, if the 2's complement of the sample time is FE0C00, then the reload value for the third timer byte would be 02, instead of FE. The time interrupt routine might then be:

```
TIMER_INTERRUPT_ROUTINE:
        DJNZ    THIRD_TIMER_BYTE,OUT
        MOV     TL0,#0
        MOV     TH0,#0CH
        MOV     THIRD_TIMERBYTE,#2
        MOV     FREQUENCY,COUNTER_LO
;Preset COUNTER to −896:
        MOV     COUNTER_LO,#80H
        MOV     COUNTER_HI,#0FCH
OUT: RETI
```

At this point the value of the frequency of the transducer signal, measured to 8-bit resolution, is contained in FREQUENCY. Note that the timer can be reloaded on the fly. Note too that for 8-bit resolution only the low byte of the frequency counter needs to be read, since the high byte is necessarily 0. However, one may want to test the high byte to ensure that it is 0, as a sanity check on the data. Both bytes, of course, must be reloaded.

PERIOD MEASUREMENTS

Measuring the period of the transducer signal means measuring the total elapsed time over a known number, N, of transducer pulses. The quantity that is directly measured is NT, where T is the period of the transducer signal in machine cycles. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{Fxtal}{F} \cdot (1/12)$$

where Fxtal is the 80C51BH clock frequency, in the same units as F.

The full scale range then is $N \cdot (Tmax - Tmin)$. For n-bit resolution

$$1 \text{ LSB} = \frac{N \cdot (Tmax - Tmin)}{2^n}$$

Therefore the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{Tmax - Tmin}$$

However, N must also be an integer. It is logical to evaluate the above formula (do not forget Tmax and Tmin have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, but it can be scaled back if desired.

For example, suppose an 8-bit resolution is wanted in the measurement of the period of a signal with a frequency that varies from 7.1 to 9 kHz. If the clock frequency is 12 MHz, Tmax is (12000 kHz/7.1 kHz) · (1/12) = 141 machine cycles. Tmin is 111 machine cycles. The required value for N, then, is 256/(141 − 111) = 8.53 periods, according to the formula. Using N = 9 periods will give a maximum NT value of 141 · 9 = 1269 machine cycles. The minimum NT will be 111 · 9 = 999 machine cycles. A lookup table can be used to scale these values back to a range of 0 to 255, giving precisely the 8-bit resolution desired.

To implement the measurement, one timer is used to measure the elapsed time, NT. The transducer is connected to one of the external interrupt pins, and this interrupt is configured to the transition-activated mode. In the transition-activated mode, every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For the specific example cited above, the interrupt routine might be:

```
INTERRUPT_RESPONSE:
        DJNZ   N,OUT
MOV     N,#9
        CLR    EA
        CLR    TR1
        MOV    NT_LO,TL1
        MOV    NT_HI,TH1
        MOV    TL1,#9
        MOV    TH1,#0
        SETB   TR1
        SETB   EA
        CALL   LOOKUP_TABLE
OUT:    RETI
```

In this routine a pulse counter N is decremented from its preset value, 9, to 0. When the counter gets to 0 it is reloaded to 9. Then all interrupts are blocked for a short time while the timer is read and cleared. The timer is stopped during the read and clear operations, so "clearing" it actually means presetting it to 9, to make up for the 9 machine cycles that are missed while the timer is stopped.

The subroutine LOOKUP_TABLE is used to scale the measurement back to the desired 8-bit resolution. It can also include built-in corrections for errors or non-linearities in the transducer's transfer function.

The subroutine uses the MOVC A, @ A + DPTR instruction to access the table, which contains 270 entries commencing at the 16-bit address referred to as TABLE. The subroutine must compute the address of the table entry that corresponds to the measured value of NT. This address is

DPTR = TABLE + NT − NTMIN,

where NTMIN = 999, in this specific example.

```
LOOKUP_TABLE:
        PUSH   ACC
        PUSH   PSW
        MOV    A,#LOW(TABLE – NTMIN)
        ADD    A,NT_LO
        MOV    DPL,A
        MOV    A,#HIGH(TABLE – NTMIN)
        ADDC   A,NT_HI
        MOV    DPH,A
        CLR    A
        MOVC   A,@A + DPTR
        MOV    PERIOD,A
        POP    PSW
        POP    ACC
        RET
```

At this point the value of the period of the transducer signal, measured to 8-bit resolution, is contained in PERIOD.

PULSE WIDTH MEASUREMENTS

The 80C51BH timers have an operating mode, called the "gate" mode, that is particularly suited to pulse-width measurements, and is useful in these applications if the transducer signal has a fixed duty cycle.

In this mode, the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low, and it can do this while the 80C51BH is in Idle. (The "gate" mode of timer operation is described in Chapter Two, Timer/Counters.) The external interrupt itself can be enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine then reads and resets the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in interrupt response time have no effect on the measurement.

Resonant transducers that are designed to fully exploit the gate mode have an internal divide-by-N circuit that fixes the duty cycle at 50% and lowers the output frequency to the range of 250 to 500 Hz (to control RFI). The transfer function between transducer period and measure and value is approximately linear, with known and repeatable error functions.

NMOS/CMOS INTERCHANGEABILITY

The CMOS version of the 8051 is architecturally identical with the NMOS version, but there are nevertheless some important differences between them of which the designer should be aware. In addition, some applications require interchangeability between NMOS and CMOS parts. The differences are as follows:

External Clock Drive: To drive the NMOS 8051 with an external clock signal, one normally grounds the XTAL1 pin and drives the XTAL2 pin. To drive the CMOS 8051 with an external clock signal, one must drive the XTAL1 pin and leave the XTAL2 pin unconnected. The reason for the difference is that in the NMOS 8051, the XTAL2 pin drives the internal clocking circuits, whereas in the CMOS version, the XTAL1 pin drives the internal clocking circuits.

There are several ways to design an external clock drive to work with both types. For low clock frequencies (below 6 MHz), the NMOS 8051 can be driven the same way as the CMOS version, namely, through XTAL1 with XTAL2 unconnected. Another way is to drive both XTAL1 and XTAL2, that is, drive XTAL1 and use an external inverter to derive from XTAL1 a signal with which to drive XTAL2.

In either case, a 74HC or 74HCT circuit makes an excellent driver for XTAL1 and/or XTAL2, because neither the NMOS nor the CMOS XTAL pins have TTL-like input logic levels.

Unused Pins: Unused pins of Ports 1, 2, and 3 can be ignored in both NMOS and CMOS designs. The internal pull-ups will put them into a defined state. Unused Port 0 pins in 8051 applications can be ignored, even if they're floating. But in 80C51BH applications, these pins should not be left afloat. They can be externally pulled up or down, or they can be internally pulled down by writing 0s to them.

80C31BH designs may or may not need pull-ups on Port 0. Pull-ups are not needed for program fetches, because in bus operations the pins are actively pulled high or low by either the 8031 or the external program memory. However, they are needed for the CMOS part if the Idle or Power-Down mode is invoked, because in these modes, Port 0 floats.

Logic Levels: If Vcc is between 4.5 V and 5.5 V, an input signal that meets the NMOS 8051 input logic levels will also meet the CMOS 80C51BH input logic levels (except for XTAL1/XTAL2 and RST). For the same Vcc condition, the CMOS device will reach or surpass the output logic levels of the NMOS device. The NMOS device will not necessarily reach the output logic levels of the CMOS device. This is an important consideration if NMOS/CMOS interchangeability must be maintained in an otherwise CMOS system.

NMOS 8051 outputs that have internal pull-ups (Ports 1, 2, and 3) "typically" reach 4 V or more if IoH is 0, but not fast enough to meet timing specs. Adding an external pull-up resistor will ensure the logic level, but still not the timing, as shown in Figure 7-16. If timing is an issue, the best way to interface NMOS to CMOS is through a 74HCT circuit.
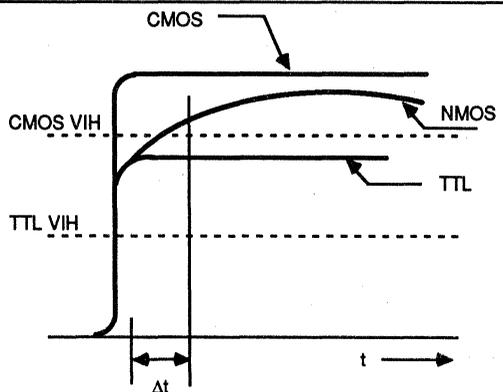
Figure 7-16. Transition Shows Unspecified Delay (Δt) in NMOS to 79HC Logic

Idle and Power Down: The Idle and Power-Down modes exist only on the CMOS devices, but if one wishes to preserve the capability of interchanging NMOS and CMOS 8051s, the software has to be designed so that the NMOS parts will respond in an acceptable manner when a CMOS reduced power mode is invoked.

For example, an instruction that invokes Power Down can be followed by a "JMP $":

```
CLR    EA
ORL    PCON,#2
JMP    $
```

The CMOS and NMOS parts will respond differently to this sequence of code. The CMOS part, going into a normal CMOS Power-Down mode, will stop fetching instructions until it gets a hardware reset. The NMOS part will go through the motions of executing the ORL instruction, and then fetch the JMP instruction. It will continue fetching and executing JMP $ until hardware reset.

Maintaining NMOS/CMOS 8051 interchangeability in response to Idle requires more planning. The NMOS part will not respond to the instruction that puts the CMOS part into Idle, so that instruction needs to be followed by a software Idle. This would be an idling loop which would be terminated by the same conditions that would terminate the CMOS hardware Idle. Then when the CMOS device goes into Idle, the NMOS version executes the idling loop until either a hardware reset or an enabled interrupt is received. Now if Idle is terminated by an interrupt, execution for the CMOS device will proceed after RETI from the instruction following the one that invoked Idle. The instruction following the one that invoked Idle is the idling loop that was inserted for the NMOS device. At this point, both the NMOS and CMOS devices must be able to fall through the loop to continue execution.

One way to achieve the desired effect is to define a "fake" Idle flag, and set it just before going into Idle. The instruction that invokes Idle is followed by a software idle:

```
SETB   IDLE
ORL    PCON,#1
JB     IDLE,$
```

Now the interrupt that terminates the CMOS Idle must also break the software idle. It does so by clearing the "Idle" bit:

```
. . .
CLR    IDLE
RETI
```

Note too that the PCON register in the NMOS 8051 contains only one bit, SMOD, whereas the PCON register in CMOS contains SMOD plus four other bits. Two of those other bits are general purpose flags. Maintaining NMOS/CMOS interchangeability requires that these flags not be used.

CHAPTER 8

80C521/80C321/80C541

CMOS Single-Chip Microcontroller

FINAL

DISTINCTIVE CHARACTERISTICS

- Software and pin-compatible with 80C51
- Dedicated Watchdog Timer
  - Robust: immune to software disables
  - Flexible: user programmable from 128 microseconds to 4 seconds at 12 MHz
- Dual Data Pointers
  - Faster external memory access
- Software Reset

	RAM (bytes)	ROM (bytes)
80C321	256	—
80C521	256	8K
80C541	256	16K

80C521 = 80C321 + 8K bytes ROM
80C541 = 80C321 + 16K bytes ROM

GENERAL DESCRIPTION

The 80C521 Family (80C521, 80C321, and 80C541) is a fully instruction-set-compatible and pin-compatible enhancement of the industry-standard 80C51 architecture. These products include a programmable Watchdog Timer and Dual Data Pointers to enhance reliability and improve performance.

The 80C521, 80C321, and 80C541 include 256 bytes of RAM. The 80C521 has 8K bytes of on-chip custom ROM, the 80C541 has 16K bytes of ROM, and the 80C321 has no on-chip ROM.

A dedicated Watchdog Timer was added to provide enhanced system reliability by increasing tolerance to noise, ESD, and software failures. This robust timing circuit has special software and electrical isolation features. For example, it cannot be disabled by potentially corrupted software.

It is user programmable from 128 microseconds to 4 seconds at 12 MHz.

The Dual Data Pointers structure speeds access to external memory by providing two identical 16-bit data pointers with a fast switching mechanism. This overcomes a traditional 8051 limitation of only a single data pointer and can improve performance of tasks such as block transfers by over 100%. For more information consult the Software Routines section in this chapter.

The 80C521 Family is offered in 40-pin plastic DIP and 44-pin PLCC packages. As with the 80C52T2/80C32T2, the PLCC package contains three additional supply connections (pins 1, 23, and 34) that greatly improve noise tolerance over packages with a single V_{CC} and V_{SS} connection.

SIMPLIFIED BLOCK DIAGRAM
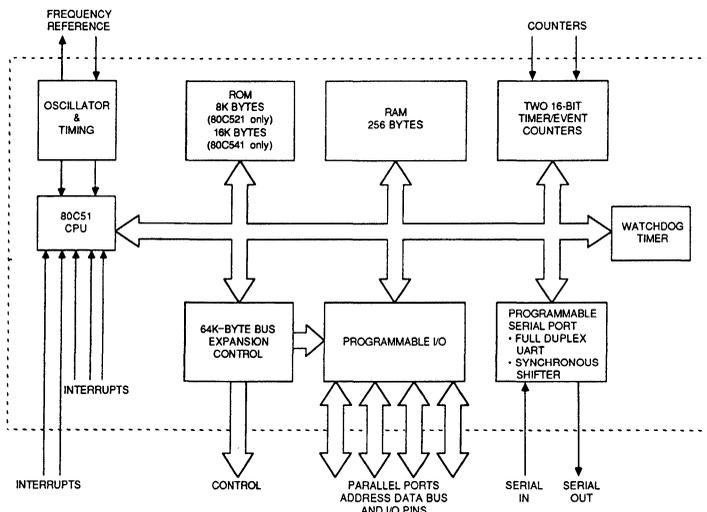
BD007216

80C521/80C321/80C541

DETAILED BLOCK DIAGRAM
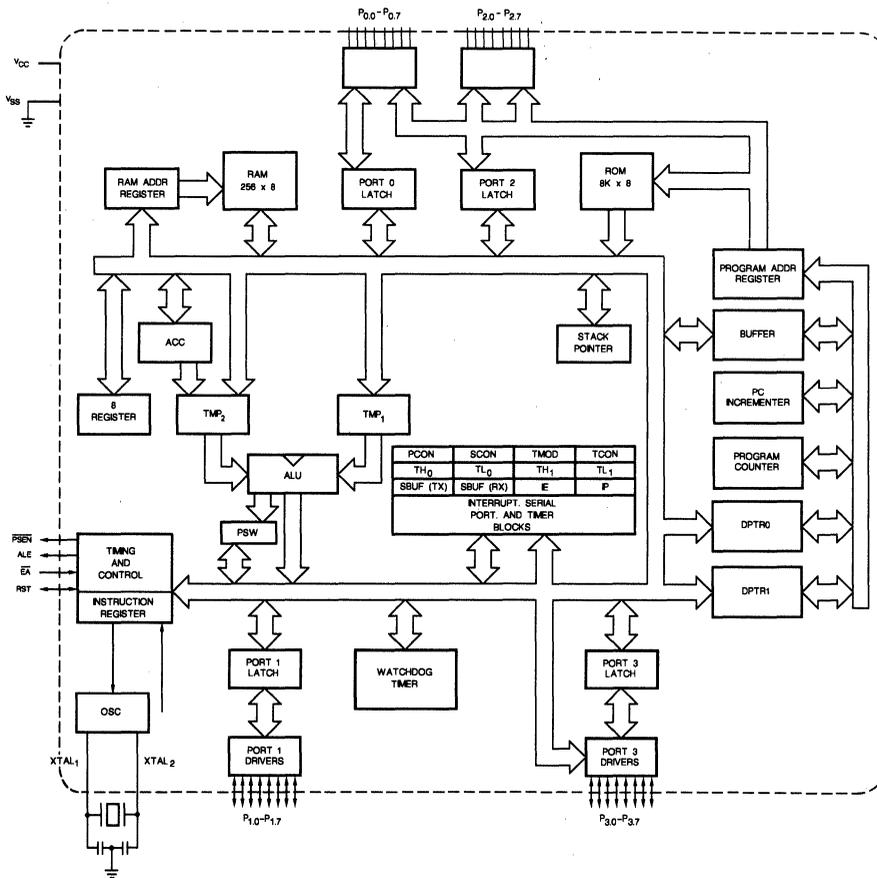
BD004097

CONNECTION DIAGRAMS
Top View

DIPs   PLCC

Pin		Pin	
P1.0	1	40	V_{CC}
P1.1	2	39	P0.0 AD_0
P1.2	3	38	P0.1 AD_1
P1.3	4	37	P0.2 AD_2
P1.4	5	36	P0.3 AD_3
P1.5	6	35	P0.4 AD_4
P1.6	7	34	P0.5 AD_5
P1.7	8	33	P0.6 AD_6
RST	9	32	P0.7 AD_7
RXD P3.0	10	31	$\overline{EA}$
TXD P3.1	11	30	ALE
$\overline{INT_0}$ P3.2	12	29	$\overline{PSEN}$
$\overline{INT_1}$ P3.3	13	28	P2.7 A_{15}
T_0 P3.4	14	27	P2.6 A_{14}
T_1 P3.5	15	26	P2.5 A_{13}
$\overline{WR}$ P3.6	16	25	P2.4 A_{12}
$\overline{RD}$ P3.7	17	24	P2.3 A_{11}
$XTAL_2$	18	23	P2.2 A_{10}
$XTAL_1$	19	22	P2.1 A_9
V_{SS}	20	21	P2.0 A_8
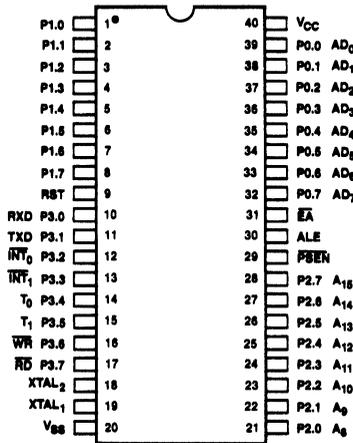
CD005554
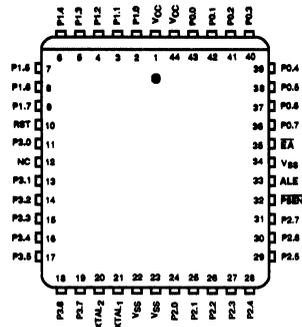
CD009444

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL
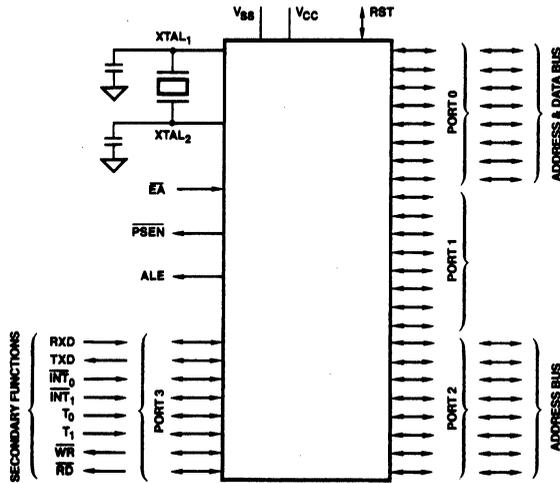
LS001324

ORDERING INFORMATION

Commodity Products

AMD commodity products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

    a. **Temperature Range**
    b. **Package Type**
    c. **Device Number**
    d. **Speed Option**
    e. **Optional Processing**

P   80C521

e. OPTIONAL PROCESSING
Blank = Standard processing

d. SPEED OPTION
Blank = 0.1 to 12 MHz
−1 = 0.1 to 16 MHz

c. DEVICE NUMBER/DESCRIPTION
80C521/80C321/80C541
CMOS Single-Chip Microcontroller

b. PACKAGE TYPE
P = 40-Pin Plastic DIP (PD 040)
N = 44-Pin Plastic Leaded Chip Carrier (PL 044)

a. TEMPERATURE RANGE
Blank = Commercial (0 to +70°C)
I = Industrial (−40 to +85°C)
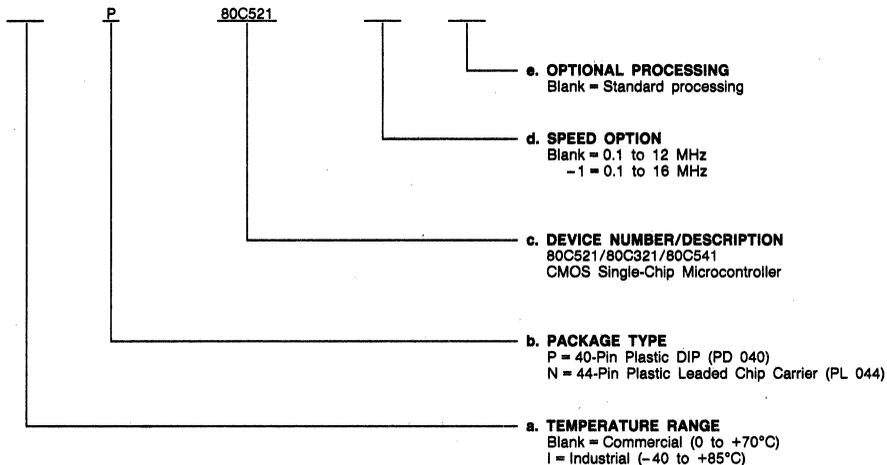
Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

Valid Combinations	
P, N IP, IN	80C521
	80C521-1
	80C321
	80C321-1
	80C541
	80C541-1

PIN DESCRIPTION

Port 0 (Bidirectional, Open Drain)

Port 0 is an open-drain bidirectional I/O port. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed Low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also outputs the code bytes during program verification in the 80C521. External pullups are required during program verification.

Port 1 (Bidirectional)

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four LSTTL inputs. Port 1 pins that have 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 1 pins that are externally being pulled Low will source current (I_{IL} on the data sheet) because of the internal pullups.

Port 1 also receives the Low-order address bytes during program verification.

Port 2 (Bidirectional)

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four LSTTL inputs. Port 2 pins having 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 2 pins externally being pulled Low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the High-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register. Port 2 also receives the High-order address bits during ROM verification.

Port 3 (Bidirectional)

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four LSTTL inputs. Port 3 pins that have 1s written to them are pulled High by the internal pullups and can be used as inputs while in this state. As inputs, Port 3 pins externally being pulled Low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features as listed below:

Port Pin	Alternate Function
$P_{3.0}$	RxD (serial input port)
$P_{3.1}$	TxD (serial output port)
$P_{3.2}$	$\overline{INT_0}$ (external interrupt 0)
$P_{3.3}$	$\overline{INT_1}$ (external interrupt 1)
$P_{3.4}$	T_0 (Timer 0 external input)
$P_{3.5}$	T_1 (Timer 1 external input)
$P_{3.6}$	$\overline{WR}$ (external Data Memory write strobe)
$P_{3.7}$	$\overline{RD}$ (external Data Memory read strobe)

RST Reset (Input/Output, Active High)

A High on this pin (for two machine cycles while the oscillator is running) resets the device. An internal diffused resistor to V_{SS} permits power-on reset, using only an external capacitor to V_{CC}.

Immediately prior to a Watchdog Reset or Software Reset, this pin is pulled High for one state time. The internal pull-up can be overdriven by an external driver capable of sinking/sourcing 2.5 mA (see Figure 6 for possible circuit configurations).

ALE Address Latch Enable (Output, Active High)

Address Latch Enable is the output pulse for latching the Low byte of the address during accesses to external memory.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, allowing use for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

$\overline{PSEN}$ Program Store Enable (Output, Active Low)

$\overline{PSEN}$ is the read strobe to external Program Memory. When the 80C521 is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external Data Memory. $\overline{PSEN}$ is not activated during fetches from internal Program Memory.

$\overline{EA}$ External Access Enable (Input, Active Low)

$\overline{EA}$ must be externally held Low to enable the device to fetch code from external Program Memory locations 0000H to 1FFFH. If $\overline{EA}$ is held High, the device executes from internal Program Memory unless the program counter contains an address greater than 1FFFH.

The 80C521 internally latches the value of the $\overline{EA}$ pin at the falling edge of the reset pulse on the RST pin during a Hardware or Power-on Reset. Once latched, the $\overline{EA}$ value cannot be changed except by a Hardware reset.

XTAL$_1$ Crystal (Input)

Input to the inverting-oscillator amplifier, and input to the internal clock-generator circuits.

XTAL$_2$ Crystal (Output)

Output from the inverting-oscillator amplifier.

V_{CC} Power Supply

Supply voltage during normal, idle, and power-down operations.

V_{SS} Circuit Ground

FUNCTIONAL DESCRIPTION

Program Memory

The 80C521 has 64K bytes of Program Memory space. The lower 8K bytes (addresses 0000H to 1FFF) may reside on-chip. Instructions residing at addresses beyond 1FFF will always be fetched externally. When the External Access (EA) pin is held Low, all code-fetch operations take place externally to the 80C521.

Data Memory

The 80C521 can address 64K bytes of Data Memory external to the chip. The MOVX instructions are used to access the external Data Memory.

The internal data memory comprises three physically distinct memory spaces. They are the lower 128 bytes of RAM, the upper 128 bytes of RAM, and the 128-byte Special Function Register (SFR) space. The lower 128 bytes of RAM can be accessed through direct addressing (i.e., MOV addr, data), or indirect addressing (i.e., MOV @ Ri). The upper 128 bytes of RAM (locations 80H through FFH) can be accessed only through indirect addressing modes. The Special Function Register space, while physically distinct from the upper 128 bytes of RAM, shares addresses with the upper 128 bytes of RAM. The SFR space may be accessed through direct addressing modes only.

The first 32 bytes of RAM contain four register banks, each of which contains eight general-purpose registers. The next 16 bytes (locations 20H through 2FH) contain 128 directly addressable bit locations. The stack may be located anywhere in the internal RAM space and may be up to 256 bytes in length.

SPECIAL FUNCTION REGISTER MAP

Addr (HEX)	Symbol	Name	Default After Power-On Reset
* 80	P0	Port 0	11111111
81	SP	Stack Pointer	00000111
82	DPL	Data Pointer Low	00000000
83	DPH	Data Pointer High	00000000
+ 84	DPL1	Data Pointer Low 1	00000000
+ 85	DPH1	Data Pointer High 1	00000000
+ 86	DPS	Data Pointer Selection	00000000
87	PCON	Power Control	0XXX0000
* 88	TCON	Timer/Counter Control	00000000
89	TMOD	Timer/Counter Mode Control	00000000
8A	TL0	Timer/Counter 0 Low Byte	00000000
8B	TL1	Timer/Counter 1 Low Byte	00000000
8C	TH0	Timer/Counter 0 High Byte	00000000
8D	TH1	Timer/Counter 1 High Byte	00000000
* 90	P1	Port 1	11111111
* 98	SCON	Serial Control	00000000
99	SBUF	Serial Data Buffer	Indeterminate
* A0	P2	Port 2	11111111
* A8	IE	Interrupt Enable Control	0XX00000
+ A9	WDS	Watchdog Selection	00000000
+ AA	WDK	Watchdog Key	00000000
* B0	P3	Port 3	11111111
* B8	IP	Interrupt Priority Control	XXX00000
* D0	PSW	Program Status Word	00000000
* E0	ACC	Accumulator	00000000
* F0	B	B Register	00000000

* Bit Addressable
+ New SFRs defined on the 80C521/80C321

Basic Timing Definitions

Instructions in the 8051 family execute in either one, two, or four machine cycles. A machine cycle comprises six state times with each state made up of two clock cycles; thus, a machine cycle lasts 12 clock cycles. With an external oscillator running at 12 MHz, a machine cycle lasts 1 μs. At 16 MHz, a machine cycle lasts 750 ns.

Reset Operation

The 80C521/80C321 may be reset by four different methods: (1) Power-On Reset, (2) Hardware Reset, (3) Watchdog Reset, and (4) Software Reset.

1. **Power-On Reset** occurs when the RST pin is wired to V_{CC} using an external capacitor, and V_{CC} is activated.

2. **Hardware Reset** occurs when the oscillator is running and the RST pin is held High for two or more machine cycles.

3. **Watchdog Reset** occurs when the count value of the Watchdog Timer is allowed to exceed the programmed value, resulting in an overflow signal that resets the chip in two machine cycles.

4. **Software Reset** occurs when the software writes a keyed sequence to the key register of the Watchdog Timer. This causes a Watchdog Reset to be immediately generated.

After Power-On Reset, the SFRs have the values indicated in the Special Function Register Map Section, and the contents of the internal RAM are undefined. Hardware Reset is the same as Power-On Reset except that the contents of the internal RAM are preserved. A Hardware Reset has priority over a Watchdog Reset or a Software Reset. The Watchdog Reset puts the 80C521 into the same state as the Hardware Reset except that the Reset Cause (RC) bit in the Watchdog Selection (WDS) register is set to a 1. The Software Reset is functionally equivalent to the Watchdog Reset.

Watchdog Timer

The Watchdog Timer (WDT) is a specially designed timer unit that will reset the chip upon reaching a pre-programmed time interval. It operates independently of the two general purpose timer/counters and is dedicated specifically to the watchdog function. The Watchdog Timer allows safe recovery from problems resulting from unexpected input conditions, external events, or programming anomalies.

The WDT is disabled following any reset. While disabled, the WDT time interval may be programmed. The WDT is enabled by a sequence of two write operations.

Once enabled, the WDT cannot be stopped (i.e., disabled) except by one of the four Reset types described in the last section. Furthermore, while the WDT is enabled, the WDT time interval cannot be modified. The WDT, however, may be cleared by software at any time with the same sequence of two write operations. The clearing operation causes the present count of the WDT to be set to zero, but it does not stop the WDT from incrementing.

If the count in the WDT ever reaches the pre-programmed value, the WDT will overflow, resetting the chip in two machine cycles. This is a Watchdog Reset. Additionally, if a system error condition is discovered, software may intentionally generate an immediate reset via the WDT, using a special sequence of write operations. This is a Software Reset.

A Watchdog Reset or Software Reset will set a special "cause" bit, allowing differentiation between these two Reset types and the Hardware or Power-On Reset types. Neither Watchdog Reset nor the Software Reset modify the contents of the internal RAM. The Watchdog Reset will cause the RST pin to be pulled High during S2P1 and S2P2 of the first cycle of the two-cycle reset, providing a hardware indication that a reset is imminent.

Two 8-bit Special Function Registers are associated with the WDT. They are as follows:

> Watchdog Selection (WDS) — Address: A9 (Hex)
>
> Watchdog Key (WDK) — Address: AA (Hex)

Watchdog Selection (WDS) — Address: A9H

The Watchdog Selection register allows the time interval of the WDT to be programmed and retains the cause of the most recent reset. This register is Read/Write, but its contents cannot be changed once the WDT has been enabled. Its default value after a Hardware or Power-On Reset = 00H. Its default value after a Watchdog Reset or Software Reset = 80H. This is the only register on the 80C521 whose initialization value differs between the two reset groups.

(MSB)							(LSB)
RC	–	TV	–	PT3	PT2	PT1	PT0
7	6	5	4	3	2	1	0

Bits 3 – 0 — Programmed Time (PT3 – PT0)

The value contained in these bits at the time the Watchdog Timer is enabled determines the time interval of the WDT. The time interval is a multiple of the input clock period. The times are decoded in the following table.

Programmable Watchdog Timing Intervals

PT3-PT0	12 MHz	16 MHz	Clock Divide Ratio
0 0000	128 μs	96 μs	1536
1 0001	256 μs	192 μs	3072
2 0010	512 μs	384 μs	6144
3 0011	1.024 ms	768 μs	12288
4 0100	2.048 ms	1.536 ms	24576
5 0101	4.096 ms	3.072 ms	49152
6 0110	8.192 ms	6.144 ms	98304
7 0111	16.384 ms	12.288 ms	196608
8 1000	32.768 ms	24.576 ms	393216
9 1001	65.536 ms	49.152 ms	786432
A 1010	131.072 ms	98.304 ms	1572864
B 1011	262.144 ms	196.608 ms	3145728
C 1100	524.288 ms	393.216 ms	6291456
D 1101	1.049 sec	786.432 ms	12582912
E 1110	2.097 sec	1.573 sec	25165824
F 1111	4.194 sec	3.146 sec	50331648

If the Programmed Time bits are read while the WDT is disabled, they will show the last value written. Once the WDT is enabled, these bits will show the programmed time of the WDT and cannot be modified.

Bit 4

Reserved. Will return an unidentified value when read.

Bit 5 — Timer Verification (TV)

This bit reflects Bit 11 of the internal counter within the Watchdog Timer. It will toggle every 4.096 ms at 12 MHz. This bit is Read-only.

Bit 6

Reserved. Will return an unidentified value when read.

Bit 7 — Reset Cause (RC)

The Reset Cause bit indicates the cause of the last reset of the 80C521. If a Power-On or Hardware Reset occurs, the bit is set to a 0 by the reset circuitry. If a Watchdog or Software Reset occurs, the bit is set to a 1 by the reset circuitry. Like the Programmed Time bits, this bit may not be modified once the WDT is enabled. Writing this bit does not affect any chip function.

Watchdog Key (WDK) — Address: AAH

This register controls the enabling and clearing of the Watchdog Timer. The writing of an A5H followed by the writing of a 5AH to this register enables the WDT to begin incrementing. It is not a requirement that the writes be on consecutive instructions, thus interrupts do not have to be disabled. Once the WDT is enabled, it may be cleared at any time by the writing of the same sequence. The clearing operation causes the present count of the WDT to be cleared, but does not stop the WDT from incrementing.

This is a Write-only register. Read operations are not defined and will not affect the WDT circuitry.

(MSB)							(LSB)
7	6	5	4	3	2	1	0

The enabling/clearing operation of the Watchdog Timer is accomplished by writing a keyed sequence of values to the WDK register. The Keyed Sequence is composed of two stages (see Figure 1).

BD007220

Figure 1. WDT Keyed Sequence Flowchart

The Keyed Sequence is in Stage 1 after all forms of reset, or following any Watchdog enable or clear operation. In Stage 1 all values written to the WDK register are ignored except A5H. An A5H causes the Keyed Sequence to enter Stage 2.

Once Stage 2 is entered, the next write to the WDK register prompts one of the following actions: (1) If the next write is again an A5H, the Keyed Sequence remains in Stage 2; (2) If the next write is a 5AH, the WDT is enabled/cleared, and the Keyed Sequence reenters Stage 1; or, (3) If the next write is any other value, a Software Reset via the WDT is generated.

Example of Write Operations to WDK:		
Write		
1st	**2nd**	**Action Taken After Second Write**
11	18	No action taken, Keyed Sequence still in Stage 1
A5	A5	Keyed Sequence enters Stage 2 and remains there
A5	5A	WDT is enabled/cleared, Sequence reenters Stage 1
A5	11	Software Reset occurs via the WDT

The two-stage feature, together with the Software Reset, greatly reduces the chance of an instruction sequence accidentally clearing the Watchdog Timer. Furthermore, while still allowing a Software Reset to be initiated, the two-stage feature reduces the chance of unintentionally generating a Software Reset.

Software Reset

A Software Reset may be accomplished through the Watchdog Timer. If an A5H is written to the Watchdog Key (WDK) register, followed by the write of a value other than A5H or 5AH, a Software Reset will be generated. This software-generated Watchdog Reset occurs regardless of whether or not the Watchdog Timer was previously enabled.

After the second value is written to the WDK register, program execution continues for one machine cycle before the reset operation begins. During S2P1 and S2P2 of this last machine cycle, the RST pin is pulled High (see Figure 6). The reset operation lasts two machine cycles and does not modify the contents of the internal RAM.

The Software Reset is functionally equivalent to the Watchdog Reset. For instance, the Reset Cause bit in WDS will be set to 1, indicating a Watchdog Reset occurred (see the Watchdog Timer section for more details).

The following code may be used to generate a Software Reset.

```
MOV WDK,#A5H ; Write A5 (Hex) to WDK

MOV WDK,#11H ; Write 11 (Hex) to WDK
                  Software Reset generated via WDT
```

Dual Data Pointers

The Dual Data Pointer structure is the means by which the 80C521 family may specify the address of an external Data Memory location. The Dual Data Pointer structure consists of two 16-bit registers that address external memory, and a single 8-bit register that allows the program code to selectively switch between them. They are located in the Special Function Register space at the following addresses:

```
82H Data Pointer Low       –(DPL)   ⎫
83H Data Pointer High      –(DPH)   ⎬ Data Pointer 0 (DPTR0)

84H Data Pointer Low 1     –(DPL1)  ⎫
85H Data Pointer High 1    –(DPH1)  ⎬ Data Pointer 1 (DPTR1)

86H Data Pointer Selection –(DPS)
```

Data Pointer 0 (DPTR0) is the original data pointer on the standard 80C51 (formerly referred to as DPTR). Data Pointer 1 (DPTR1) is an additional data pointer with identical characteristics. Instructions that refer to DPTR refer to the data pointer that is currently selected in the Data Pointer Selection (DPS) register. The six instructions that reference DPTR are as follows:

Instruction	Description
INC DPTR	;Increments the data pointer by 1
MOV DPTR, #data16	;Loads DPTR with a 16-bit constant
MOVC A, @A + DPTR	;Move code byte relative to DPTR to Acc
MOVX A, @DPTR	;Move external RAM (16-bit address) to Acc
MOVX @DPTR, A	;Move Acc to external RAM (16-bit address)
JMP @A + DPTR	;Jump indirect relative to DPTR

It is also possible to access each data pointer on a byte-by-byte basis by specifying its low or high byte in an instruction that accesses the Special Function Registers. These instructions can be executed at any time regardless of which of the two data pointers is currently selected. Three examples are as follows:

Instruction	Description
MOV DPH,R3	;Move the contents of Register 3 into DPH
MOV A,DPL1	;Move the contents of DPL1 into the Acc
PUSH DPH1	;Push the contents of DPH1 onto the stack

The Dual Data Pointer structure saves both time and code space by eliminating the need for frequent loading and unloading of a single data pointer. For instance, block move operations in external memory can be more efficiently implemented by using DPTR0 as the source address, and DPTR1 as the destination address. The Dual Data Pointer structure enhances this operation considerably.

Data Pointer Selection (DPS) — Address: 86H

This register determines which of the two data pointers is currently selected. Once a data pointer is selected, the six DPTR instructions refer only and always to that data pointer until another data pointer is selected. Upon reset, the default data pointer (DPTR0) will be selected, thus retaining compatibility with existing 8051-family devices. The switch between data pointers may be accomplished with a single cycle instruction (such as: INC DPS or MOV DPS,A). The default value at reset = 00H. This is a Read/Write register.

(MSB)							(LSB)
0	0	0	0	0	0	0	SEL0
7	6	5	4	3	2	1	0
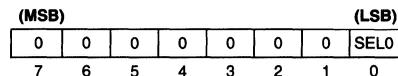
Bit 0 — Select 0 (SEL0)

If this bit is 0, the original data pointer, DPTR0, is selected. If this bit is 1, DPTR1 is selected. This bit may be written by software at any time. When read, its current value is presented.

Bits 7 – 1

Reserved. Will return 0 when read.

Data Pointer Low (DPL) — Address: 82H

DPL is a Read/Write register that contains the low byte of Data Pointer 0. It may be accessed at any time with an instruction that specifies a direct byte as a source of destination. However, SEL0 in the DPS register must be set to 0 before any of the six explicit DPTR instructions will access this register. The default at reset = 00H.

(MSB)							(LSB)
7	6	5	4	3	2	1	0
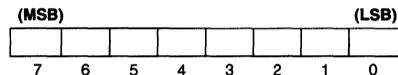
Data Pointer High (DPH) — Address: 83H

DPH is a Read/Write register that contains the high byte of Data Pointer 0. It may be accessed at any time with an instruction that specifies a direct byte as a source or destination. However, SEL0 in the DPS register must be set to 0 before any of the six explicit DPTR instructions will access this register. The default at reset = 00H.

(MSB)							(LSB)
7	6	5	4	3	2	1	0
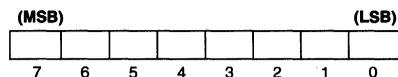
Data Pointer Low 1 (DPL1) — Address: 84H

DPL1 is a Read/Write register that contains the low byte of Data Pointer 1. It may be accessed at any time with an instruction that specifies a direct byte as a source or destination. However, SEL0 in the DPS register must be set to 1 before any of the six explicit DPTR instructions will access this register. The default at reset = 00H.

(MSB)   (LSB)

7	6	5	4	3	2	1	0
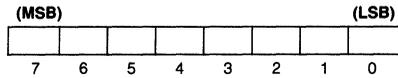
Data Pointer High 1 (DPH1) — Address: 85H

DPH1 is a Read/Write register that contains the high byte of Data Pointer 1. It may be accessed at any time with an instruction that specifies a direct byte as a source or destination. However, SEL0 in the DPS register must be set to 1 before any of the six explicit DPTR instructions will access this register. The default at reset = 00H.

(MSB)   (LSB)

7	6	5	4	3	2	1	0
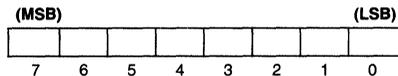
Dual Data Pointer Example

To load both data pointers after reset:

Method 1:

```
MOV DPL ,#data8      ;load low byte of DPTR0
MOV DPH ,#data8      ;load high byte of DPTR0
MOV DPL1,#data8      ;load low byte of DPTR1
MOV DPH1,#data8      ;load high byte of DPTR1
   (Data Pointer 0 is still selected.)
```

Method 2:

```
MOV                  ;load DPTR0 with 16-bit const.
DPTR,#data16
INC DPS              ;switch data pointers
MOV                  ;load DPTR1 with 16-bit const.
DPTR,#data16
   (Data Pointer 1 is now selected.)
```

Oscillator Characteristics

XTAL$_1$ and XTAL$_2$ are the input and output, respectively, of an inverting amplifier which is configured for use as an on-chip oscillator (see Figure 2). Either a quartz crystal or ceramic resonator may be used.

To drive the device from an external clock source, XTAL$_1$ should be driven while XTAL$_2$ is left unconnected (see Figure 3). There are no requirements on the duty cycle of the external clock signal since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum High and Low times specified on the data sheet must be observed.

TC003411
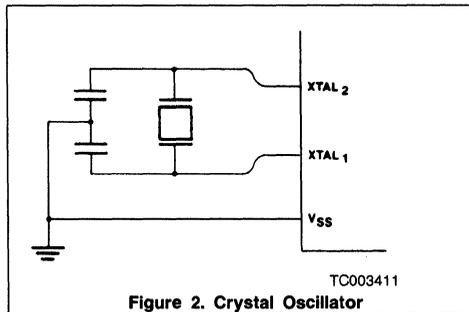
Figure 2. Crystal Oscillator

TC003392
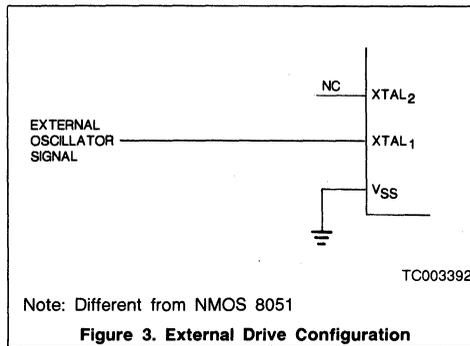
Note: Different from NMOS 8051

Figure 3. External Drive Configuration

Idle and Power-Down Operation

Figure 4 shows the internal operation of the Idle and Power-Down circuitry. Power-Down operation disconnects the clock source from all internal chip circuitry. Idle mode operation allows the interrupt, serial port, timers, and watchdog circuitry to continue to function while the CPU is stopped. If the Watchdog Timer is enabled, Power-Down operation is not possible.

These special modes are activated by software via the Special Function Register, PCON (Table 1). Its hardware address is 87H; PCON is not bit-addressable.

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is 0XXX0000.

TC003382
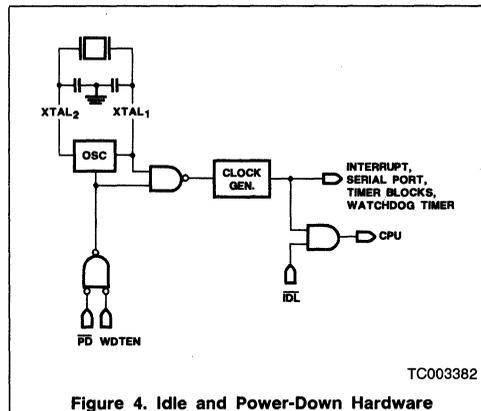
Figure 4. Idle and Power-Down Hardware

TABLE 1. PCON (Power Control Register)

(MSB) (LSB)

SMOD	–	–	–	GF1	GF0	PD	IDL

Symbol	Position	Name and Description
SMOD	PCON.7	Double-baud-rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2, or 3.
–	PCON.6	(Reserved)
–	PCON.5	(Reserved)
–	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit
GF0	PCON.2	General-purpose flag bit
PD	PCON.1	Power-Down bit. Setting this bit activates power-down operation.
IDL	PCON.0	Idle-mode bit. Setting this bit activates idle-mode operation.

Idle Mode

The instruction that sets PCON.0 is the last instruction executed in the normal operating mode before the Idle mode is activated. Once in the Idle mode, the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all other registers in the 80C521 maintain their data during Idle. Table 2 describes the status of the external pins during Idle mode.

There are three possible ways to terminate the Idle mode. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt is serviced, and following RETI, the next instruction to be executed will be the one following the instruction that wrote a 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an enabled interrupt, the service routine can examine the status of the flag bits.

The second way of terminating the Idle mode is with a Hardware Reset.

The third way of terminating the Idle mode is with the Watchdog Timer. If the WDT is not enabled, then it has no effect on subsequent Idle mode operations. If the WDT is enabled before Idle mode is entered, it will continue to increment in the normal fashion. If the WDT overflows, the 80C521 will experience a Watchdog Reset and Idle mode will be terminated. If Idle mode is terminated by any method other than a reset, the Watchdog Timer will continue to run.

Power-Down Mode

The instruction that sets PCON.1 is the last executed prior to going into Power-Down. Once in Power-Down, the oscillator is stopped. The contents of the on-chip RAM are preserved. The Special Function Registers are saved until a Hardware Reset is generated. A hardware reset is the only way of exiting the Power-Down mode.

Power-Down mode cannot be entered while the Watchdog Timer is enabled. If a write of the value 1 is attempted into the PD bit of the PCON register, its value will remain 0, and no Power-Down operation will take place. To enter Power-Down mode, the Watchdog Timer must first be disabled via a Hardware Reset, Software Reset, or Watchdog Reset. After reset, the Watchdog Timer is disabled, allowing Power-Down mode to be entered.

In the Power-Down mode, V_{CC} may be lowered to minimize circuit power consumption. Care must be taken to ensure the voltage is not reduced until the Power-Down mode is entered, and that the voltage is restored before the Hardware Reset is applied. Hardware Reset frees the oscillator and should not be released until the oscillator has restarted and stabilized.

Table 2 describes the status of the external pins while in the Power-Down mode. It should be noted that if the Power-Down mode is activated while in external program memory, the port data that is held in the Special Function Register P_2 is restored to Port 2. If the data is a 1, the port pin is held High during the Power-Down mode by the strong pullup, P_1, shown in Figure 5.

80C521 I/O Ports

The I/O port drive of the 80C521 is similar to the 8051. The I/O buffers for Ports 1, 2, and 3 are implemented as shown in Figure 5.

When the port latch contains a 0, all pFETS in Figure 5 are off while the nFET is turned on. When the port latch makes a 0-to-1 transition, the nFET turns off. The strong pullup pFET, P_1, turns on for two oscillator periods, pulling the output High very rapidly. As the output line is drawn High, pFET P_3 turns on through the inverter to supply the I_{OH} source current. This inverter and P_3 form a latch that holds the 1 and is supported by P_2.

When Port 2 is used as an address port, for access to external program or data memory, any address bit that contains a 1 will have its strong pullup turned on for the entire duration of the external memory access.

When an I/O pin on Ports 1, 2, or 3 is used as an input, the user should be aware that the external circuit must sink current during the logical 1-to-0 transition. The maximum sink current is specified as I_{TL} under the D.C. specifications. When the input goes below approximately 2 V, P_3 turns off to save I_{CC} current. Note, when returning to a logical 1, P_2 is the only internal pullup that is on. This will result in a slow rise time if the user's circuit does not force the input line High.

TABLE 2. STATUS OF THE EXTERNAL PINS DURING IDLE AND POWER-DOWN MODES

Mode	Program Memory	ALE	PSEN	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Port Data	Port Data	Port Data	Port Data
Idle	External	1	1	Floating	Port Data	Address	Port Data
Power-Down	Internal	0	0	Port Data	Port Data	Port Data	Port Data
Power-Down	External	0	0	Floating	Port Data	Port Data	Port Data

Figure 5. I/O Buffers in the 80C521 (Ports 1, 2, 3)
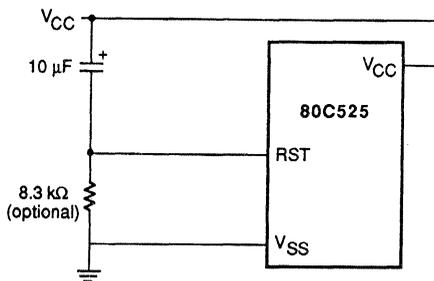
Standard (80C51) Reset Circuit

Neither a Watchdog nor a Software Reset will affect the Standard reset circuitry, nor can they be sensed by the Standard (80C51) reset circuitry.
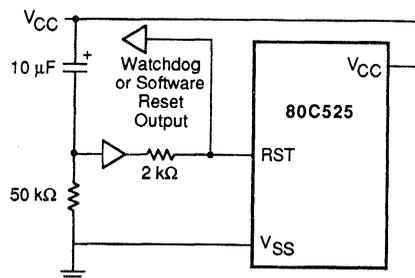
Watchdog Reset Circuit

The reset circuit shown above may be used to sense a Watchdog or Software Reset. For $V_{CC} = 5$ V, the driver output must be able to source/sink 2.5 mA.

Figure 6. RESET Configurations

ABSOLUTE MAXIMUM RATINGS

Storage Temperature .... −65°C to +150°C
Voltage on Any
Pin to V_{SS} .... −0.5 V to V_{CC} +0.5 V
Voltage on V_{CC} to V_{SS} .... −0.5 V to 6.5 V
Power Dissipation .... 200 mW

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
Temperature (T_A) .... 0 to +70°C
Supply Voltage (V_{CC}) .... +4.5 V to +5.5 V
Ground (V_{SS}) .... 0 V
Industrial (I) Devices
Temperature (T_A) .... −40 to +85°C
Supply Voltage (V_{CC}) .... +4.5 V to +5.5 V
Ground (V_{SS}) .... 0 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
V_{IL}	Input Low Voltage (Except $\overline{EA}$)		−0.5	0.2 V_{CC} − 0.1	V
V_{IL1}	Input Low Voltage ($\overline{EA}$)		−0.5	0.2 V_{CC} − 0.3	V
V_{IH}	Input High Voltage (Except XTAL$_1$, RST)		0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Input High Voltage (XTAL$_1$ RST)		0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Output Low Voltage (Ports 1, 2, 3)	I_{OL} = 1.6 mA (Note 1)		0.45	V
V_{OL1}	Output Low Voltage (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA (Note 1)		0.45	V
V_{OH}	Output High Voltage (Ports 1, 2, 3)	I_{OH} = −60 μA, V_{CC} = 5 V±10%	2.4		V
		I_{OH} = −25 μA	0.75 V_{CC}		V
		I_{OH} = −10 μA	0.9 V_{CC}		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode, ALE $\overline{PSEN}$)	I_{OH} = −800 μA, V_{CC} = 5 V±10%	2.4		V
		I_{OH} = −300 μA	0.75 V_{CC}		V
		I_{OH} = −80 μA (Note 2)	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3)	V_{IN} = 0.45 V		−50	μA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1, 2, 3)	V_{IN} = 2 V		−650	μA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V_{IN} < V_{CC}		±10	μA
RRST	Reset Pulldown Resistor		50	150	kΩ
CIO	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{PD}	Power-Down Current	V_{CC} = 2 to 6 V (Note 3)		50	μA

MAXIMUM I_{CC} (mA)

	Operating (Note 4)			Idle (Note 5)		
Freq. V_{CC}	4.5 V	5 V	5.5 V	4.5 V	5 V	5.5 V
0.1 MHz	2.2	3.1	3.8	0.7	0.9	1.4
3.5 MHz	6	8	10	1.5	2	3
8.0 MHz	11	14	18	2.5	3.5	5
12 MHz	15	20	25	3.5	5	6
16 MHz	19	25	32	4.5	6.5	8.5

Notes: 1. Capacitive loading on ports may cause spurious noise pulses to be superimposed on the V_{OL}S of ALE and other ports. The noise is due to external bus capacitance discharging into the port pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8 V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt-Trigger STROBE input. This note pertains to dual-in-line packages only. The additional V_{CC} and V_{SS} connections on the PLCC package from AMD removes this design consideration.
2. Capacitive loading on ports may cause the V_{OH} on ALE and $\overline{PSEN}$ to momentarily fall below the 0.9 V_{CC} specification when the address bits are stabilizing. This note pertains to dual-in-line packages only. The additional V_{CC} and V_{SS} connections on the PLCC package from AMD remove this design consideration.
3. Power-Down I_{CC} is measured with all output pins disconnected: $\overline{EA}$ = Port 0 = V_{CC}; XTAL$_2$ NC; RST = V_{SS}.
4. I_{CC} is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{SS} + 0.5 V, V_{IH} = V_{CC} − 0.5 V; XTAL$_2$ NC; $\overline{EA}$ = RST = Port 0 = V_{CC}. Typical values are approximately 50% lower. I_{CC} would be slightly higher if a crystal oscillator was used.
5. Idle I_{CC} is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{SS} + 0.5 V, V_{IH} = V_{CC} − 0.5 V; XTAL$_2$ NC; Port 0 = V_{CC}; $\overline{EA}$ = RST = V_{SS}, and the Watchdog Timer disabled.

SWITCHING CHARACTERISTICS over operating range (C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF; C_L for All Other Outputs = 80 pF)

Parameter Symbol	Parameter Description	16-MHz Osc.		12-MHz Osc.		Variable Oscillator		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS								
1/TCLCL	Oscillator Frequency					0.1	16	MHz
TLHLL	ALE Pulse Width	85		127		2TCLCL – 40		ns
TAVLL	Address Valid to ALE Low	7		28		TCLCL – 55		ns
TLLAX	Address Hold After ALE Low	27		48		TCLCL – 35		ns
TLLIV	ALE Low to Valid Instr. In		150		234		4TCLCL – 100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	22		43		TCLCL – 40		ns
TPLPH	$\overline{PSEN}$ Pulse Width	142		205		3TCLCL – 45		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr. In		83		145		3TCLCL – 105	ns
TPXIX	Input Instr. Hold After $\overline{PSEN}$	0		0		0		ns
TPXIZ	Input Instr. Float After $\overline{PSEN}$		38		59		TCLCL – 25	ns
TAVIV	Address to Valid Instr. In		208		312		5TCLCL – 105	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		10		10		10	ns
TRLRH	$\overline{RD}$ Pulse Width	275		400		6TCLCL – 100		ns
TWLWH	$\overline{WR}$ Pulse Width	275		400		6TCLCL – 100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		148		252		5TCLCL – 165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		0		ns
TRHDZ	Data Float After $\overline{RD}$		55		97		2TCLCL – 70	ns
TLLDV	ALE Low to Valid Data In		350		517		8TCLCL – 150	ns
TAVDV	Address to Valid Data In		398		585		9TCLCL – 165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	137	238	200	300	3TCLCL – 50	3TCLCL + 50	ns
TAVWL	Address Valid to Read or Write Low	120		203		4TCLCL – 130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	2		23		TCLCL – 60		ns
TQVWH	Valid Data to Write High	287		433		7TCLCL – 150		ns
TWHQX	Data Hold After $\overline{WR}$	12		33		TCLCL – 50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	22	103	43	123	TCLCL – 40	TCLCL + 40	ns

SWITCHING WAVEFORMS

External Program Memory Read Cycle

WF021962

External Data Memory Read Cycle

WF020962

SWITCHING WAVEFORMS (continued)

External Data Memory Write Cycle

WF020932

Shift Register Timing Waveforms

WF020951

80C521/80C321/80C541

EXTERNAL CLOCK DRIVE

Parameter Symbol	Parameter Description	Min.	Max.	Unit
1/TCLCL	Oscillator Frequency	0.1	16	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

External Clock Drive Waveform

WF020911

SERIAL PORT TIMING—SHIFT REGISTER MODE
Test Conditions: T_A = 0°C to 70°C; V_{CC} = 5 V ±10%; V_{SS} = 0 V; Load Capacitance = 80 pF

Parameter Symbol	Parameter Description	16-MHz Osc.		Variable Oscillator		Unit
		Min.	Max.	Min.	Max.	
TXLXL	Serial Port Clock Cycle Time	750		12TCLCL		ns
TQVXH	Output Data Setup to Clock Rising Edge	492		10TCLCL – 133		ns
TXHQX	Output Data Hold After Clock Rising Edge	8		2TCLCL – 117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		492		10TCLCL – 133	ns

AC Testing

WF020901

AC inputs during testing are driven at V_{CC} – 0.5 for a logic 1 and 0.45 V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Input/Output Waveform

WF020941

For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, and begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs. $I_{OL}/I_{OH} \geq \pm 20$ mA.

Float Waveform

CLOCK WAVEFORMS

INTERNAL CLOCK

| STATE 4 | STATE 5 | STATE 6 | STATE 1 | STATE 2 | STATE 3 | STATE 4 | STATE 5 |
| P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 |

XTAL 2

ALE

THESE SIGNALS ARE NOT ACTIVATED DURING THE EXECUTION OF A MOVX INSTRUCTION

EXTERNAL PROGRAM MEMORY FETCH

$\overline{PSEN}$

$AD_0 - AD_7$

DATA — PCL OUT — DATA — PCL OUT — DATA — PCL OUT

SAMPLED FLOAT

$A_8 - A_{15}$

INDICATES ADDRESS TRANSITIONS

READ CYCLE

$\overline{RD}$

PCL OUT (IS PROGRAM MEMORY IS EXTERNAL)

$AD_0 - AD_7$

DPL OR RI OUT — DATA

SAMPLED FLOAT

$A_8 - A_{15}$

WRITE CYCLE

$\overline{WR}$

PCL OUT (EVEN IF PROGRAM MEMORY IS INTERNAL)

$AD_0 - AD_7$

DPL OR RI OUT — DATA OUT

PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)

$A_8 - A_{15}$

PORT OPERATION
MOV PORT, SRC

OLD DATA | NEW DATA

MOV DEST, P1
(INCLUDES INT0, INT1, T0, T1)

P1 PIN SAMPLED

P1 PIN SAMPLED

SERIAL PORT SHIFT CLOCK

TXD (MODE 0)

RXD SAMPLED

RXD SAMPLED

WF020923

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though (T_A = 25°C, fully loaded), $\overline{RD}$ and $\overline{WR}$ propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

80C521/80C321/80C541

TABLE 3. 80C521/80C321/80C541 INSTRUCTION SET

Instructions That Affect Flag Setting*

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C, bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C, bit	X		
RLC	X			CJNE	X		
SETB C	1						

Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request and push the PC; to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (2.25 to 5.25 μs at 16 MHz).

*Note that operations on SFR byte address D0H or bit addresses D0 – D7H (i.e., the PSW or bits in the PSW) will also affect flag settings.

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Move immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Mode direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Move 16-bit constant to Data Pointer	3	2
MOVC	A,@A + DPTR	Move Code byte relative to DPTR to Accumulator	1	2
MOVC	A,@A + PC	Move Code byte relative to PC to Accumulator	1	2
MOVX	A,@Ri	Move External RAM (8-bit address) to Accumulator	1	2
MOVX	A,@DPTR	Move External RAM (16-bit address) to Accumulator	1	2
MOVX	@Ri,A	Move Accumulator to External RAM (8-bit address)	1	2
MOVX	@DPTR,A	Move Accumulator to External RAM (16-bit address)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte off of stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	1
XCHD	A,@Ri	Exchange indirect RAM's least sig nibble with A's LSN	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry Flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry Flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement Carry Flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry Flag	2	2
ANL	C,/bit	AND complement of direct bit to Carry	2	2
ORL	C,bit	OR direct bit to Carry Flag	2	2
ORL	C,/bit	OR complement of direct bit to Carry	2	2
MOV	C,bit	Move direct bit to Carry Flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

LOGIC

Mnemonic		Description	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1

LOGIC (Continued)

Mnemonic		Description	Byte	Cyc
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	1
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate Accumulator Left through Carry Flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate Accumulator Right through Carry Flag	1	1
SWAP	A	Exchange nibbles within the Accumulator	1	1

ARITHMETIC

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with carry	1	1
ADDC	A,direct	Add direct byte to Accumulator with Carry Flag	2	1
ADDC	A,@Ri	Add indirect RAM and Carry Flag to Accumulator	1	1
ADDC	A,#data	Add immediate data and Carry Flag to Accumulator	2	1
SUBB	A,Rn	Subtract register from Accumulator with Borrow	1	1
SUBB	A,direct	Subtract direct byte from Accumulator with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from Accumulator with Borrow	1	1
SUBB	A,#data	Subtract immediate data from Accumulator with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
MUL	AB	Multiply Accumulator times B	1	4
DIV	AB	Divide Accumulator by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

OTHER			
Mnemonic	Description	Byte	Cyc
NOP	No Operation	1	1

CONTROL TRANSFER (BRANCH)

Mnemonic	Description	Byte	Cyc
AJMP addr11	Absolute Jump	2	2
LJMP addr16	Long Jump	3	2
SJMP rel	Short Jump (relative addr)	2	2
JMP @A + DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if Accumulator is zero	2	2
JNZ rel	Jump if Accumulator is not zero	2	2
JC rel	Jump if Carry Flag is set	2	2
JNC rel	Jump if carry is not set	2	2
JB bit,rel	Jump relative if direct bit is set	3	2
JNB bit,rel	Jump relative if direct bit is not set	3	2
JBC bit,rel	Jump relative if direct bit is set, then clear bit	3	2
CJNE A,direct,rel	Compare direct byte to Accumulator and Jump if not Equal	3	2
CJNE A,#data,rel	Compare immediate to Accumulator and Jump if not Equal	3	2
CJNE Rn,#data,rel	Compare immediate to reg and Jump if not Equal	3	2
CJNE @Ri,#data,rel	Compare immediate to indirect RAM and Jump if not Equal	3	2
DJNZ Rn,rel	Decrement register and Jump if not zero	2	2
DJNZ direct,rel	Decrement direct byte and Jump if not zero	3	2

CONTROL TRANSFER (SUBROUTINE)

Mnemonic	Description	Byte	Cyc
ACALL addr11	Absolute Subroutine Call	2	2
LCALL addr16	Long Subroutine Call	3	2
RET	Return from Subroutine Call	1	2
RETI	Return from Interrupt Call	1	2

Notes on Data Addressing Modes:

Rn   –Working register R0–R7 of the currently selected Register bank.

direct   –128 internal RAM locations, any I/O port, control, or Special Function Registers.

@Ri   –Indirect internal RAM location addressed by register R0 or R1.

#data   –8-bit constant included in instruction.

#data16   –16-bit constant included as bytes 2 and 3 of instruction.

bit   –128 software flags, any I/O pin, control, or status bit.

Notes on Program Addressing Modes:

addr16   –Destination address for LCALL and LJMP may be anywhere within the 64-kilobyte program memory address space.

addr11   –Destination address for ACALL and AJMP will be within the same 2-kilobyte page of program memory as the first byte of the following instruction.

rel   –SJMP and all conditional jumps include as 8-bit offset by Range is +127, –128 bytes relative to first byte of the following instruction.

TABLE 4. INSTRUCTION OPCODES IN HEXADECIMAL ORDER

Hex Code	Bytes	Mnemonic	Operands	Hex Code	Bytes	Mnemonic	Operands
00	1	NOP		29	1	ADD	A,R1
01	2	AJMP	Code addr	2A	1	ADD	A,R2
02	3	LJMP	Code addr	2B	1	ADD	A,R3
03	1	RR	A	2C	1	ADD	A,R4
04	1	INC	A	2D	1	ADD	A,R5
05	2	INC	Data addr	2E	1	ADD	A,R6
06	1	INC	@R0	2F	1	ADD	A,R7
07	1	INC	@R1	30	3	JNB	Bit addr,code addr
08	1	INC	R0	31	2	ACALL	Code addr
09	1	INC	R1	32	1	RETI	
0A	1	INC	R2	33	1	RLC	A
0B	1	INC	R3	34	2	ADDC	A,#data
0C	1	INC	R4	35	2	ADDC	A,data addr
0D	1	INC	R5	36	1	ADDC	A,@R0
0E	1	INC	R6	37	1	ADDC	A,@R1
0F	1	INC	R7	38	1	ADDC	A,R0
10	3	JBC	Bit addr,code addr	39	1	ADDC	A,R1
11	2	ACALL	Code addr	3A	1	ADDC	A,R2
12	3	LCALL	Code addr	3B	1	ADDC	A,R3
13	1	RRC	A	3C	1	ADDC	A,R4
14	1	DEC	A	3D	1	ADDC	A,R5
15	2	DEC	Data addr	3E	1	ADDC	A,R6
16	1	DEC	@R0	3F	1	ADDC	A,R7
17	1	DEC	@R1	40	2	JC	Code addr
18	1	DEC	R0	41	2	AJMP	Code addr
19	1	DEC	R1	42	2	ORL	Data addr,A
1A	1	DEC	R2	43	3	ORL	Data addr,#data
1B	1	DEC	R3	44	2	ORL	A,#data
1C	1	DEC	R4	45	2	ORL	A,data addr
1D	1	DEC	R5	46	1	ORL	A,@R0
1E	1	DEC	R6	47	1	ORL	A,@R1
1F	1	DEC	R7	48	1	ORL	A,R0
20	3	JB	Bit addr,code addr	49	1	ORL	A,R1
21	2	AJMP	Code addr	4A	1	ORL	A,R2
22	1	RET		4B	1	ORL	A,R3
23	1	RL	A	4C	1	ORL	A,R4
24	2	ADD	A,#data	4D	1	ORL	A,R5
25	2	ADD	A,data addr	4E	1	ORL	A,R6
26	1	ADD	A,@R0	4F	1	ORL	A,R7
27	1	ADD	A,@R1	50	2	JNC	Code addr
28	1	ADD	A,R0	51	2	ACALL	Code addr

Hex Code	Bytes	Mnemonic	Operands	Hex Code	Bytes	Mnemonic	Operands
52	2	ANL	Data addr,A	AA	2	MOV	R2,data addr
53	3	ANL	Data addr,#data	AB	2	MOV	R3,data addr
54	2	ANL	A,#data	AC	2	MOV	R4,data addr
55	2	ANL	A,data addr	AD	2	MOV	R5,data addr
56	1	ANL	A,@R0	AE	2	MOV	R6,data addr
57	1	ANL	A,@R1	AF	2	MOV	R7,data addr
58	1	ANL	A,R0	B0	2	ANL	C,/bit addr
59	1	ANL	A,R1	B1	2	ACALL	Code addr
5A	1	ANL	A,R2	B2	2	CPL	Bit addr
5B	1	ANL	A,R3	B3	1	CPL	C
5C	1	ANL	A,R4	B4	3	CJNE	A,#data,code addr
5D	1	ANL	A,R5	B5	3	CJNE	A,data addr,code addr
5E	1	ANL	A,R6	B6	3	CJNE	@R0,#data,code addr
5F	1	ANL	A,R7				
60	2	JZ	Code addr	B7	3	CJNE	@R1,#data,code addr
61	2	AJMP	Code addr				
62	2	XRL	Data addr,A	B8	3	CJNE	R0,#data,code addr
63	3	XRL	Data addr,#data	B9	3	CJNE	R1,#data,code addr
64	2	XRL	A,#data	BA	3	CJNE	R2,#data,code addr
65	2	XRL	A,data addr	BB	3	CJNE	R3,#data,code addr
66	1	XRL	A,@R0	BC	3	CJNE	R4,#data,code addr
67	1	XRL	A,@R1	BD	3	CJNE	R5,#data,code addr
68	1	XRL	A,R0	BE	3	CJNE	R6,#data,code addr
69	1	XRL	A,R1	BF	3	CJNE	R7,#data,code addr
6A	1	XRL	A,R2	C0	2	PUSH	Data addr
6B	1	XRL	A,R3	C1	2	AJMP	Code addr
6C	1	XRL	A,R4	C2	2	CLR	Bit addr
6D	1	XRL	A,R5	C3	1	CLR	C
6E	1	XRL	A,R6	C4	1	SWAP	A
6F	1	XRL	A,R7	C5	2	XCH	A,data addr
70	2	JNZ	Code addr	C6	1	XCH	A,@R0
71	2	ACALL	Code addr	C7	1	XCH	A,@R1
72	2	ORL	C,bit addr	C8	1	XCH	A,R0
73	1	JMP	@A +DPTR	C9	1	XCH	A,R1
74	2	MOV	A,#data	CA	1	XCH	A,R2
75	3	MOV	Data addr,#data	CB	1	XCH	A,R3
76	2	MOV	@R0,#data	CC	1	XCH	A,R4
77	2	MOV	@R1,#data	CD	1	XCH	A,R5
78	2	MOV	R0,#data	CE	1	XCH	A,R6
79	2	MOV	R1,#data	CF	1	XCH	A,R7
7A	2	MOV	R2,#data	D0	2	POP	Data addr
7B	2	MOV	R3,#data	D1	2	ACALL	Code addr
7C	2	MOV	R4,#data	D2	2	SETB	Bit addr
7D	2	MOV	R5,#data	D3	1	SETB	C
7E	2	MOV	R6,#data	D4	1	DA	A
7F	2	MOV	R7,#data	D5	3	DJNZ	Data addr,code addr
80	2	SJMP	Code addr	D6	1	XCHD	A,@R0
81	2	AJMP	Code addr	D7	1	XCHD	A,@R1
82	2	ANL	C,bit addr	D8	2	DJNZ	R0,code addr
83	1	MOVC	A,@A +PC	D9	2	DJNZ	R1,code addr
84	1	DIV	AB	DA	2	DJNZ	R2,code addr
85	3	MOV	Data addr,data addr	DB	2	DJNZ	R3,code addr
86	2	MOV	Data addr,@R0	DC	2	DJNZ	R4,code addr
87	2	MOV	Data addr,@R1	DD	2	DJNZ	R5,code addr
88	2	MOV	Data addr,R0	DE	2	DJNZ	R6,code addr
89	2	MOV	Data addr,R1	DF	2	DJNZ	R7,code addr
8A	2	MOV	Data addr,R2	E0	1	MOVX	A,@DPTR
8B	2	MOV	Data addr,R3	E1	2	AJMP	Code addr
8C	2	MOV	Data addr,R4	E2	1	MOVX	A,@R0
8D	2	MOV	Data addr,R5	E3	1	MOVX	A,@R1
8E	2	MOV	Data addr,R6	E4	1	CLR	A
8F	2	MOV	Data addr,R7	E5	2	MOV	A,data addr
90	3	MOV	DPTR,#data	E6	1	MOV	A,@R0
91	2	ACALL	Code addr	E7	1	MOV	A,@R1
92	2	MOV	Bit addr,C	E8	1	MOV	A,R0
93	1	MOVC	A,@A +DPTR	E9	1	MOV	A,R1
94	2	SUBB	A,#data	EA	1	MOV	A,R2
95	2	SUBB	A,data addr	EB	1	MOV	A,R3
96	1	SUBB	A,@R0	EC	1	MOV	A,R4
97	1	SUBB	A,@R1	ED	1	MOV	A,R5
98	1	SUBB	A,R0	EE	1	MOV	A,R6
99	1	SUBB	A,R1	EF	1	MOV	A,R7
9A	1	SUBB	A,R2	F0	1	MOVX	@DPTR,A
9B	1	SUBB	A,R3	F1	2	ACALL	Code addr
9C	1	SUBB	A,R4	F2	1	MOVX	@R0,A
9D	1	SUBB	A,R5	F3	1	MOVX	@R1,A
9E	1	SUBB	A,R6	F4	1	CPL	A
9F	1	SUBB	A,R7	F5	2	MOV	Data addr,A
A0	2	ORL	C,/bit addr	F6	1	MOV	@R0,A
A1	2	AJMP	Code addr	F7	1	MOV	@R1,A
A2	2	MOV	C,bit addr	F8	1	MOV	R0,A
A3	1	INC	DPTR	F9	1	MOV	R1,A
A4	1	MUL	AB	FA	1	MOV	R2,A
A5		Reserved		FB	1	MOV	R3,A
A6	2	MOV	@R0,data addr	FC	1	MOV	R4,A
A7	2	MOV	@R1,data addr	FD	1	MOV	R5,A
A8	2	MOV	R0,data addr	FE	1	MOV	R6,A
A9	2	MOV	R1,data addr	FF	1	MOV	R7,A

87C521/87C541

CMOS Single-Chip Microcontrollers

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- Software and pin-compatible with 80C51, 80C521, and 80C541
- Beneficial for prototyping and initial production
- All 80C521 and 80C541 features retained
- Flashrite^{TM} EPROM programming
- Two-level Program Memory Lock
- 32-Byte Encryption Array
- In-Circuit Test Mode facilitates testing

	RAM (bytes)	EPROM (bytes)
87C521	256	8K
87C541	256	16K

87C521 = User programmable 80C521
87C541 = User programmable 80C541

GENERAL DESCRIPTION

The 87C521 and 87C541 are CMOS EPROM versions of the 80C521 and 80C541, respectively. The 87C521 includes 8K bytes of on-chip EPROM, and the 87C541 includes 16K bytes of EPROM.

These user-programmable products are software- and pin-compatible with their ROM-based counterparts. All of the 80C521 and 80C541 features are retained, including a robust Watchdog Timer, Dual Data Pointers, and Software Reset. For more information consult the 80C521/80C541 Datasheet (order #09136).

The EPROM features on the 87C51 and 87C52T2 have also been retained. A two-level programmable lock structure prevents externally fetched code from accessing internal Program Memory and can disable EPROM verification and programming. A 32-byte Encryption Array can be used to encode the program code bytes during EPROM verification. A Flashrite programming algorithm allows the 8K-byte 87C521 and 16K-byte 87C541 to be programmed in approximately 24 and 48 seconds, respectively.

BLOCK DIAGRAM

BD007750

87C521/87C541

Publication #	Rev.	Amendment
09744	B	/0

Issue Date: **October 1989**

CONNECTION DIAGRAMS
Top View

DIP

P1.0	1	40	V_{CC}
P1.1	2	39	P0.0 AD_0
P1.2	3	38	P0.1 AD_1
P1.3	4	37	P0.2 AD_2
P1.4	5	36	P0.3 AD_3
P1.5	6	35	P0.4 AD_4
P1.6	7	34	P0.5 AD_5
P1.7	8	33	P0.6 AD_6
RST	9	32	P0.7 AD_7
RXD P3.0	10	31	$\overline{EA}/V_{PP}$
TXD P3.1	11	30	ALE/$\overline{PROG}$
$\overline{INT_0}$ P3.2	12	29	$\overline{PSEN}$
$\overline{INT_1}$ P3.3	13	28	P2.7 A_{15}
T_0 P3.4	14	27	P2.6 A_{14}
T_1 P3.5	15	26	P2.5 A_{13}
$\overline{WR}$ P3.6	16	25	P2.4 A_{12}
$\overline{RD}$ P3.7	17	24	P2.3 A_{11}
$XTAL_2$	18	23	P2.2 A_{10}
$XTAL_1$	19	22	P2.1 A_9
V_{SS}	20	21	P2.0 A_8
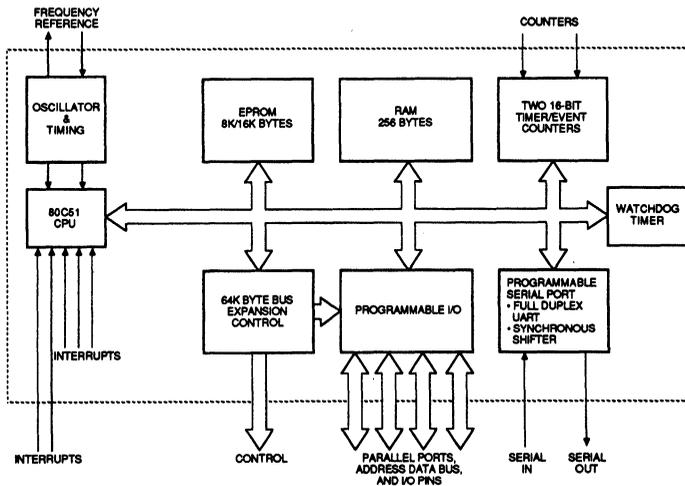
CD005552

LCC
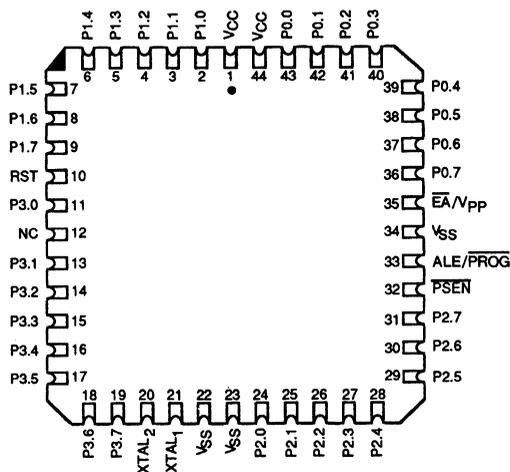
CD010872

PLCC
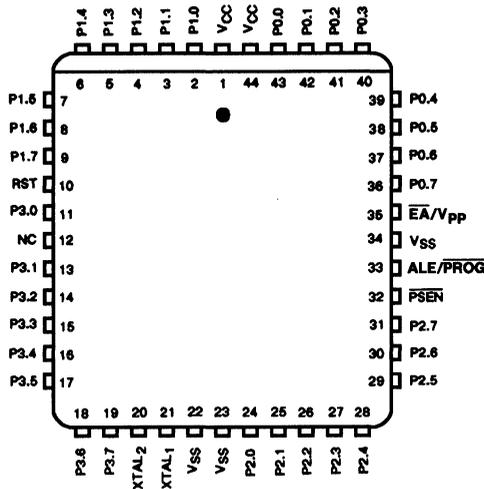
CD009442

Note: Pin 1 is marked for orientation.
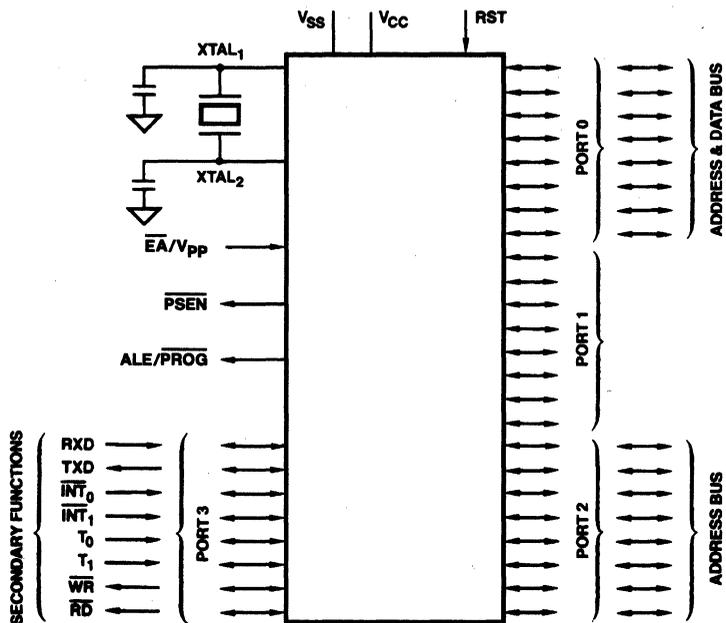
LOGIC SYMBOL

LS001326

ORDERING INFORMATION

Commodity Products

AMD commodity products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

a. **Temperature Range**
b. **Package Type**
c. **Device Number**
d. **Speed Option**
e. **Optional Processing**
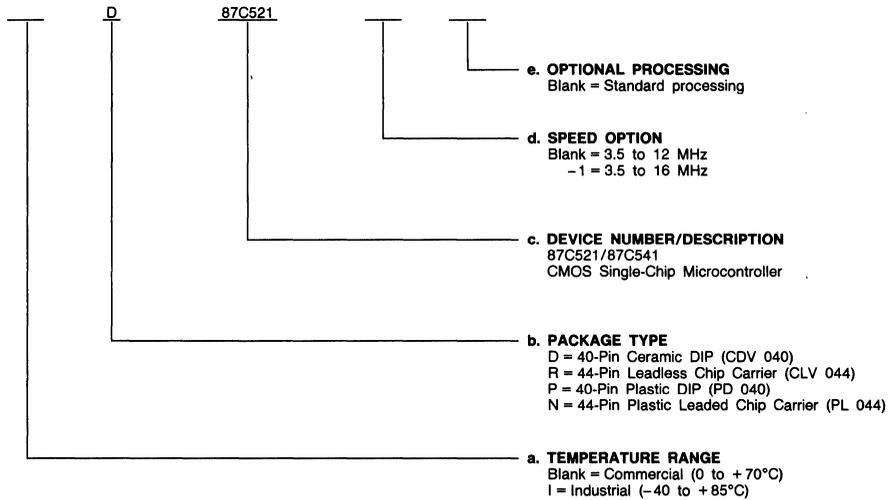
<u>D</u>   87C521

e. OPTIONAL PROCESSING
Blank = Standard processing

d. SPEED OPTION
Blank = 3.5 to 12 MHz
−1 = 3.5 to 16 MHz

c. DEVICE NUMBER/DESCRIPTION
87C521/87C541
CMOS Single-Chip Microcontroller

b. PACKAGE TYPE
D = 40-Pin Ceramic DIP (CDV 040)
R = 44-Pin Leadless Chip Carrier (CLV 044)
P = 40-Pin Plastic DIP (PD 040)
N = 44-Pin Plastic Leaded Chip Carrier (PL 044)

a. TEMPERATURE RANGE
Blank = Commercial (0 to +70°C)
I = Industrial (−40 to +85°C)

Valid Combinations	
	87C521
D, R, P, N	87C521-1
ID, IR, IP, IN	87C541
	87C541-1

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

Port 0   (Bidirectional; Open Drain)

Port 0 is an open-drain I/O port. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also outputs the code bytes during program verification in the 87C521/87C541. External pullups are required during program verification.

Port 1   (Bidirectional)

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four LS TTL inputs. Port 1 pins that have 1s written to them are pulled High by the internal pullups and—while in this state—can be used as inputs. As inputs, Port 1 pins that are externally being pulled Low will source current (I_{IL} on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification.

Port 2   (Bidirectional)

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four LS TTL inputs. Port 2 pins having 1s written to them are pulled High by the internal pullups and—while in this state—can be used as inputs. As inputs, Port 2 pins externally being pulled Low will source current (I_{IL}) because of internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function register.

Port 2 also receives the high-order address bits during the programming of the EPROM and during program verification of the EPROM, as well as some control signals.

Port 3   (Bidirectional)

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four LS TTL inputs. Port 3 pins having 1s written to them are pulled High by the internal pullups and—while in this state—can be used as inputs. As inputs, Port 3 pins externally being pulled Low will source current (I_{IL}) because of the pullups. Port 3 also receives some control signals for EPROM programming and program verification.

Port 3 also serves the functions of various special features as listed below:

Port Pin	Alternate Function
$P_{3.0}$	RxD (Serial Input Port)
$P_{3.1}$	TxD (Serial Output Port)
$P_{3.2}$	$\overline{INT}_0$ (External Interrupt 0)
$P_{3.3}$	$\overline{INT}_1$ (External Interrupt 1)
$P_{3.4}$	T_0 (Timer 0 External Input)
$P_{3.5}$	T_1 (Timer 1 External Input)
$P_{3.6}$	$\overline{WR}$ (External Data Memory Write Strobe)
$P_{3.7}$	$\overline{RD}$ (External Data Memory Read Strobe)

RST   Reset (Input; Active High)

This pin is used to reset the device when held High for two machine cycles while the oscillator is running. A small internal resistor permits power-on reset using only a capacitor connected to V_{CC}.

Immediately prior to a Watchdog Reset or Software Reset, this pin is pulled High for one state time. The internal pullup can be overdriven by an external driver capable of sinking/sourcing 2.5 mA. (See Figure 6 of the 80C521 Datasheet, order #09136C/0, for possible circuit configurations.)

ALE/$\overline{PROG}$ Address Latch Enable/Program Pulse (Input/Output)

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE can drive eight LS TTL inputs.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, allowing use for external-timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory. This pin also accepts the program pulse input ($\overline{PROG}$) when programming the EPROM.

$\overline{PSEN}$ Program Store Enable (Output; Active Low)

$\overline{PSEN}$ is the read strobe to external Program Memory. $\overline{PSEN}$ can drive eight LS TTL inputs. When the device is executing code from an external program memory, $\overline{PSEN}$ is activated twice each machine cycle—except that two $\overline{PSEN}$ activations are skipped during each access to external Data Memory. $\overline{PSEN}$ is not activated during fetches from internal Program Memory.

$\overline{EA}$/V_{PP} External Access Enable/Programming Voltage (Input; Active Low)

$\overline{EA}$ must be externally held Low to enable the device to fetch code from external Program Memory locations 0000H to 1FFFH for the 87C521 and 3FFFH for the 87C541. If $\overline{EA}$ is held High, the 87C521/87C541 executes from internal Program Memory unless the program counter exceeds 1FFFH and 3FFFH respectively.

This pin also receives the 12.75-V programming supply voltage during programming of the EPROM.

XTAL$_1$ Crystal (Input)

Input to the inverting-oscillator amplifier, and input to the internal clock-generator circuits.

XTAL$_2$ Crystal (Output)

Output of the inverting-oscillator amplifier.

V_{CC} Power Supply

Power supply during normal, idle, and power-down operations.

V_{SS} Circuit Ground

PROGRAMMING

The 87C521/87C541 can be programmed with the Flashrite algorithm. It differs from other methods in the value used for Vpp (programming supply voltage) and in the width and number of the ALE/PROG pulses.

To program the EPROM, either the internal or external oscillator must be running between 4 and 6 MHz, since the internal bus is used to transfer address and program data to the appropriate internal registers. Table 1 shows the various EPROM programming modes.

Table 1. EPROM Programming Modes for the 87C521/87C541

Mode	RST	PSEN	ALE/PROG	EA/Vpp	P2.7	P2.6	P3.7	P3.6
Program Code	H	L	L*	Vpp	H	L	H	H
Verify Code	H	L	H	Vppx	L	L	H	H
Pgm Encryption Table	H	L	L*	Vpp	H	L	H	L
Pgm Lock Bit 1	H	L	L*	Vpp	H	H	H	H
Pgm Lock Bit 2	H	L	L*	Vpp	H	H	L	L
Read Silicon Signature	H	L	H	H	L	L	L	L

Key: H = Logic High for that pin
L = Logic Low for that pin
Vpp = 12.75 V ±0.25 V
Vcc = 5 V ±10% during programming and verification
2.0 V < Vppx < 13.0 V

*ALE/PROG receives 25 programming pulses while Vpp is held at 12.75 V. Each programming pulse is Low for 100 μs (±10% μs) and High for a minimum of 10 μs.

Programming

The programming configuration for the 87C521 is shown in Figure 1. The address of the EPROM location to be programmed is applied to Ports 1 and 2 as shown in the figure. The programming configuration of the 87C541 is identical except that P2.5 is also used as an address input. The code byte to be programmed into that location is applied to Port 0. Once RST, PSEN, Port 2, and Port 3 are held to the levels indicated in Figure 1, ALE/PROG is pulsed Low 25 times, as shown in Figure 2.

The maximum voltage applied to the EA/Vpp pin must not exceed 13 V at any time as specified for Vpp. Even a slight spike can cause permanent damage to the device. The Vpp source should thus be well-regulated and glitch-free.

When programming, a 0.1-μF capacitor is required across Vpp and ground to suppress spurious transients that may damage the device.
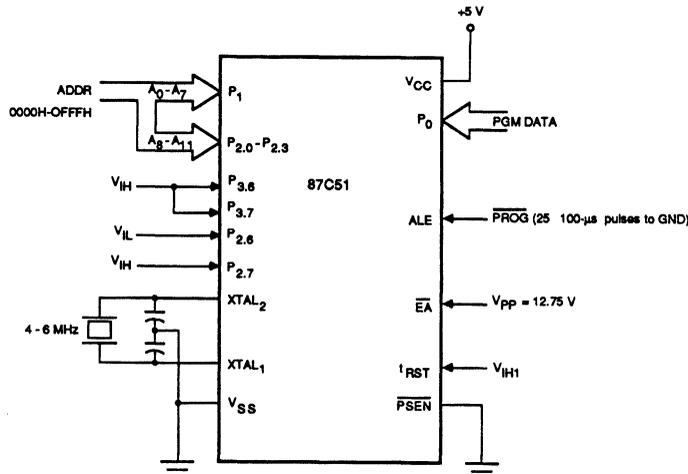
TC004691

Figure 1. 87C521 Programming Configuration
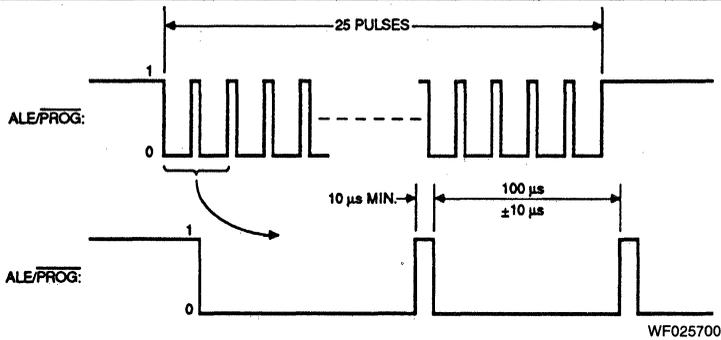
Figure 2. $\overline{\text{PROG}}$ Waveforms

Program Verification

The 87C521/87C541 provides a method of reading the programmed code bytes in the EPROM array for program verification. This function is possible as long as Lock Bit 2 has not been programmed.

For program verification, the address of the Program Memory location to be read is applied to Ports 1 and 2 as shown in Figure 3. Verification of the 87C541 is identical except that P2.5 is also used as an address input. Once RST, $\overline{\text{PSEN}}$, Port 2, and Port 3 are held to the levels indicated, the contents of the addressed location will be emitted on Port 0. External pullups are required on Port 0 for this operation. The EPROM programming and verification waveforms provide further details.
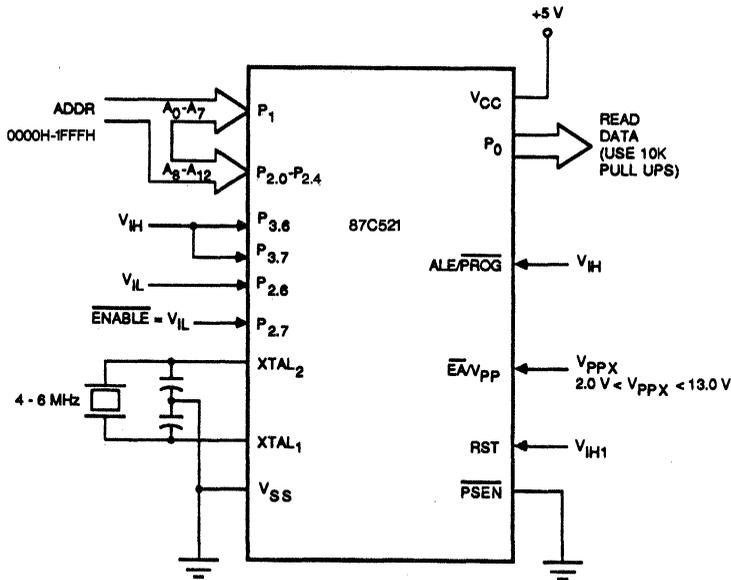
Figure 3. 87C521 Program Verification

Program Encryption Table

The 87C521/87C541 features a 32-byte Encryption Array. It can be programmed by the customer, thus encrypting the program code bytes read during EPROM verification. The EPROM verification procedure is performed as usual except that each code byte comes out logically X-NORed with one of the 32 key bytes.

The key byte used is the one whose address corresponds to the lower 5 bits of the EPROM verification address. Thus, when the EPROM is verified starting with address 0000H, all 32 keys in their correct sequence must be known. Unprogrammed bytes have the value FFH. Thus, if the Encryption Table is left unprogrammed, no encryption will be performed, since any byte X-NORed with FFH leaves that byte unchanged.

To program the Encryption Table, programming is set up as usual, except that P3.6 is held Low, as shown in Table 1. The 25-pulse programming sequence is applied to each address, 00 through 1FH. The programming of these bytes does not affect the standard 4K-byte EPROM array. When the Encryption Table is programmed, the Program Verify operation will produce only encrypted data.

The Encryption Table cannot be directly read. The programming of Lock Bit 1 will disable further Encryption Table programming.

Security Lock Bits

The 87C521/87C541 contains two Lock Bits that can be programmed to obtain additional security features. P = Programmed and U = Unprogrammed.

Lock Bit 1	Lock Bit 2	Result
U	U	Normal Operation
P	U	• Externally fetched code cannot access internal Program Memory • All further Programming disabled (except Lock Bit 2)
U	P	Reserved
P	P	• Externally fetched code cannot access internal Program Memory • All further Programming disabled • Program Verification disabled

To program the Lock Bits, a 100 pulse programming sequence is required using the levels shown in Table 1. After Lock Bit 1 is programmed, further programming of the Code Memory and Encryption Table is disabled. However, Lock Bit 2 may still be programmed, providing the highest level of security available on the 87C521/87C541.

Silicon Signature Verification

AMD supports silicon signature verification for the 87C521/87C541. The manufacturer code and part code can be read from the device before any programming is done to enable the EPROM Programmer to recognize the device.

To read the silicon signature, the external pins are set up as shown in Figure 4. This procedure is the same as a normal verification except that P3.6 and P3.7 are pulled to a logic Low. The values returned are:

Manufacturer Code	Address: 0030H	Code: 01H
Part Code: 87C521	Address: 0031H	Code: 32H
Part Code: 87C541	Address: 0031H	Code: 32H

Code 01H indicates AMD as the manufacturer. Code 32H indicates that the device type is the 87C521 or 87C541.

In-Circuit Test Mode

The In-Circuit Test Mode facilitates testing and debugging of systems using the 87C521/87C541 without the device having to be removed from the circuit. The In-Circuit Test Mode is invoked by:

1. Pulling ALE Low while RST is held High and $\overline{PSEN}$ is High.

2. Holding ALE Low as RST is deactivated.

While the device is in In-Circuit Test Mode, the Port 0 pins go into a float state, and the other port pins and ALE and $\overline{PSEN}$ are weakly pulled High. The oscillator circuit remains active. While the 87C521/87C541 is in this mode, an emulator or test CPU can be used to drive the circuit. Normal operation is restored when a Hardware Reset is applied.

Erasure Characteristics

Light and other forms of electromagnetic radiation can lead to erasure of the EPROM when exposed for extended periods of time.

Wavelengths of light shorter than 4000 angstroms, such as sunlight or indoor fluorescent lighting, can eventually cause inadvertent erasure and, therefore, should not be allowed to expose the EPROM for lengthy durations (approximately one week in sunlight or three years in room-level fluorescent lighting). It is suggested that the window be covered with an opaque label if an application is likely to subject the device to this type of radiation.

It is recommended that ultraviolet light (of 2537 angstroms) be used at a dose of at least 15 W-sec/cm^2 when erasing the EPROM. An ultraviolet lamp rated at 12,000 μW/cm^2 held one inch away for 20–30 minutes should be sufficient.

EPROM erasure leaves the Program Memory in an "all ones" state.
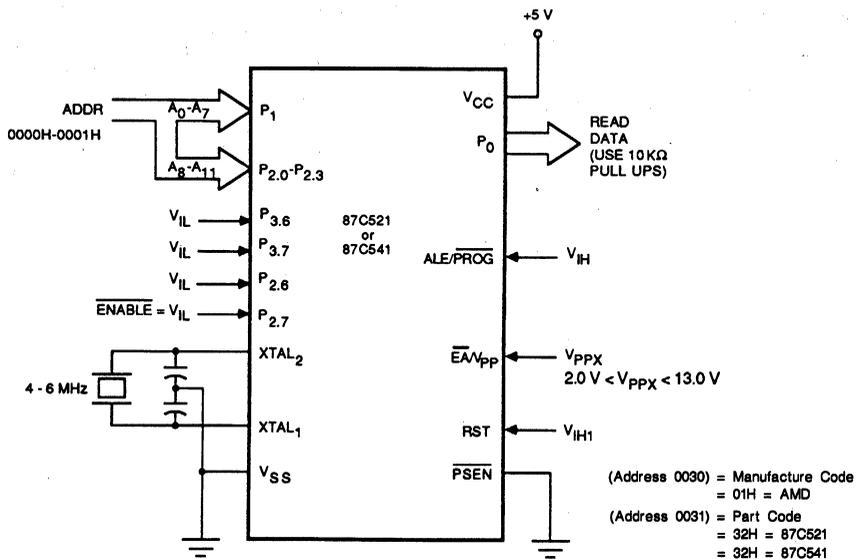
Figure 4. 87C521/87C541 Silicon Signature Verification Configuration
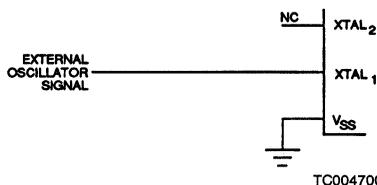
Oscillator Characteristics

$XTAL_1$ and $XTAL_2$ are the input and output, respectively, of an inverting amplifier that is configured for use as an on-chip oscillator (see Figure 5). Either a quartz crystal or ceramic resonator may be used.

To drive the device from an external clock source, $XTAL_1$ should be driven while $XTAL_2$ is left unconnected (see Figure 6). There are no requirements on the duty cycle of the external clock signal since the input to the internal clocking circuitry is through a divide-by-two flip-flop; but minimum and maximum High and Low times specified on the data sheet must be observed.

Figure 5. Crystal Oscillator

Figure 6. External Drive Configuration

ABSOLUTE MAXIMUM RATINGS

Storage Temperature .... −65 to +150°C
Voltage on EA/VPP Pin to VSS .... −0.5 to +13.0 V
Voltage on VCC to VSS .... −0.5 to +6.5 V
Voltage on Any Other Pin to VSS .... −0.5 to +6.5 V
Power Dissipation ....200 mW

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
    Ambient Temperature (T$_A$) .... 0 to +70°C
    Supply Voltage (V$_{CC}$) .... +4.5 to +5.5 V
    Ground (V$_{SS}$) .... 0 V

Industrial (I) Devices
    Ambient Temperature (T$_A$) .... −40 to +85°C
    Supply Voltage (V$_{CC}$) .... +4.5 to +5.5 V
    Ground (V$_{SS}$) .... 0 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating ranges

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
V$_{IL}$	Input Low Voltage (Except EA)		−0.5	0.2 V$_{CC}$ − 0.1	V
V$_{IL1}$	Input Low Voltage (EA)		0	0.2 V$_{CC}$ − 0.3	V
V$_{IH}$	Input High Voltage (Except XTAL$_1$, RST)		0.2 V$_{CC}$ + 0.9	V$_{CC}$ + 0.5	V
V$_{IH1}$	Input High Voltage to XTAL$_1$, RST		0.7 V$_{CC}$	V$_{CC}$ + 0.5	V
V$_{OL}$	Output Low Voltage (Ports 1, 2, 3)	I$_{OL}$ = 1.6 mA (Note 1)		0.45	V
V$_{OL1}$	Output Low Voltage (Port 0, ALE, PSEN)	I$_{OL}$ = 3.2 mA (Note 1)		0.45	V
V$_{OH}$	Output High Voltage (Ports 1, 2, 3), ALE, PSEN	I$_{OH}$ = −60 μA, V$_{CC}$ = 5 V ±10%	2.4		V
		I$_{OH}$ = −10 μA	0.9 V$_{CC}$		
V$_{OH1}$	Output High Voltage (Port 0 in External Bus Mode)	I$_{OH}$ = −800 μA, V$_{CC}$ = 5 V ±10%	2.4		V
		I$_{OH}$ = −80 μA (Note 2)	0.9 V$_{CC}$		
I$_{IL}$	Logical 0 Input Current (Ports 1, 2, 3)	V$_{IN}$ = 0.45 V		−50	μA
I$_{TL}$	Logical 1-to-0 Transition Current (Ports 1, 2, 3)	(Note 3)		−650	μA
I$_{LI}$	Input Leakage Current (Port 0)	V$_{IN}$ = V$_{IL}$ or V$_{IH}$		±10	μA
I$_{CC}$	Power Supply Current: Active Mode @ 12 MHz (Note 4)	(Note 5)		Note 4	mA
	Idle Mode @ 12 MHz (Note 4)			Note 4	
	Power-Down Mode			50	μA
RRST	Reset Pulldown Resistor		50	300	kΩ
C$_{IO}$	Pin Capacitance	Test Freq = 1 MHz, T$_A$ = 25°C		10	pF

Notes: 1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V$_{OL}$s of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8 V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
2. Capacitive loading on Ports 0 and 2 may cause the V$_{OH}$ on ALE and PSEN to momentarily fall below the 0.9 V$_{CC}$ specification when the address bits are stabilizing.
3. Pins of Ports 1, 2, and 3 source a transition current when they are being externally driven from 1 to 0. The transition current reaches its maximum value when V$_{IN}$ is approximately 2 V.
4. I$_{CCMAX}$ at other frequencies is given by:
    Active Mode: I$_{CC}$ TYPICAL = 0.94 x Freq + 13.71   I$_{CCMAX}$ = 1.38 x Freq + 20.4
    Idle Mode: I$_{CC}$ TYPICAL = 0.38 x Freq + 5.4   I$_{CCMAX}$ = 0.38 x Freq + 11.9
    where Freq is the external oscillator frequency in MHz. I$_{CCMAX}$ is given in mA.
5. Active Mode I$_{CC}$ is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V$_{IL}$ = V$_{SS}$ + 0.5 V, V$_{IH}$ = V$_{CC}$ − 0.5 V; XTAL$_2$ NC; EA = RST = Port 0.
    Idle Mode I$_{CC}$ is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V$_{IL}$ = V$_{SS}$ + 0.5 V, V$_{IH}$ = V$_{CC}$ − 0.5 V; XTAL$_2$ = NC; Port 0 = V$_{CC}$; EA = RST = V$_{SS}$.
    Power-Down Mode I$_{CC}$ is measured with all output pins disconnected; EA = Port 0 = V$_{CC}$; XTAL$_2$ NC; RST = V$_{SS}$.

SWITCHING CHARACTERISTICS over operating ranges
(Load Capacitance for Port 0, ALE, and $\overline{PSEN}$ = 100 pF, Load Capacitance for All Other Outputs = 80 pF)

Parameter Symbol	Parameter Description	16 MHz Osc. Min.	16 MHz Osc. Max.	12 MHz Osc. Min.	12 MHz Osc. Max.	Variable Oscillator Min.	Variable Oscillator Max.	Unit
1/TCLCL	Oscillator Frequency					3.5	16	MHz
TLHLL	ALE Pulse Width	85		127		2TCLCL–40		ns
TAVLL	Address Valid to ALE Low	7		28		TCLCL–55		ns
TLLAX	Address Hold After ALE Low	27		48		TCLCL–35		ns
TLLIV	ALE Low to Valid Instr. In		150		234		4TCLCL–100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	22		43		TCLCL–40		ns
TPLPH	$\overline{PSEN}$ Pulse Width	142		205		3TCLCL–45		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr. In		83		145		3TCLCL–105	ns
TPXIX	Input Instr. Hold After $\overline{PSEN}$	0		0		0		ns
TPXIZ	Input Instr. Float After $\overline{PSEN}$		38		59		TCLCL–25	ns
TAVIV	Address to Valid Instr. In		208		312		5TCLCL–105	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		10		10		10	ns
TRLRH	$\overline{RD}$ Pulse Width	275		400		6TCLCL–100		ns
TWLWH	$\overline{WR}$ Pulse Width	275		400		6TCLCL–100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		148		252		5TCLCL–165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		0		ns
TRHDZ	Data Float After $\overline{RD}$		55		97		2TCLCL–70	ns
TLLDV	ALE Low to Valid Data In		350		517		8TCLCL–150	ns
TAVDV	Address to Valid Data In		398		585		9TCLCL–165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	137	238	200	300	3TCLCL–50	3TCLCL+50	ns
TAVWL	Address Valid to $\overline{RD}$ or $\overline{WR}$ Low	120		203		4TCLCL–130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	2		23		TCLCL–60		ns
TQVWH	Data Valid to $\overline{WR}$ High	287		433		7TCLCL–150		ns
TWHQX	Data Hold After $\overline{WR}$	12		33		TCLCL–50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	22	103	43	123	TCLCL–40	TCLCL+40	ns

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
———	MUST BE STEADY	WILL BE STEADY
⑂⑂⑂	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
⑂⑂⑂	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
⑂⑂⑂	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
⑂⑂⑂	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

87C521/87C541

SWITCHING WAVEFORMS

External Program Memory Read Cycle

WF021962

External Data Memory Read Cycle

WF020963

SWITCHING WAVEFORMS (continued)

WF020934

External Data Memory Write Cycle

WF020951

Shift Register Timing Waveforms

EXTERNAL CLOCK DRIVE

Parameter Symbol	Parameter Description	Min.	Max.	Unit
1/TCLCL	Oscillator Frequency	3.5	16	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

External Clock Drive Waveform

WF020910

SERIAL PORT TIMING — SHIFT REGISTER MODE

(Test Conditions: T_A = 0 to +70°C; V_{CC} = 5 V ±10%; V_{SS} = 0 V; Load Capacitance = 80 pF)

Parameter Symbol	Parameter Description	16 MHz Osc.		Variable Oscillator		Unit
		Min.	Max.	Min.	Max.	
TXLXL	Serial Port Clock Cycle Time	750		12TCLCL		ns
TQVXH	Output Data Setup to Clock Rising Edge	492		10TCLCL–133		ns
TXHQX	Output Data Hold After Clock Rising Edge	8		2TCLCL–117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL–133	ns

AC Testing

WF020900

AC inputs during testing are driven at V_{CC} – 0.5 for a logic 1 and 0.45 V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Input/Output Waveform

WF020940

For timing purposes a port pin is no longer floating when a 100-mV change from load voltage occurs, and begins to float when a 100-mV change from the loaded V_{OH}/V_{OL} level occurs. $I_{OL}/I_{OH} \geq \pm 20$ mA.

Float Waveform

EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS
(T_A = +21 to +27°C)

Parameter Symbol	Parameter Description	Min.	Max.	Unit
V_{PP}	Programming Supply Voltage	12.5	13.0	V
I_{PP}	Programming Supply Current		50	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{PROG}$	48TCLCL		
TGHAX	Address Hold After $\overline{PROG}$	48TCLCL		
TDVGL	Data Setup to $\overline{PROG}$	48TCLCL		
TGHDX	Data Hold After $\overline{PROG}$	48TCLCL		
TEHSH	$P_{2.7}$ ($\overline{ENABLE}$) High to V_{PP}	48TCLCL		
TSHGL	V_{PP} Setup to $\overline{PROG}$	10		μs
TGHSL	V_{PP} Hold After $\overline{PROG}$	10		μs
TGLGH	$\overline{PROG}$ Width	90	110	μs
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{ENABLE}$ to Data Valid		48TCLCL	
TEHQZ	Data Float After $\overline{ENABLE}$	0	48TCLCL	
TGHGL	$\overline{PROG}$ High to $\overline{PROG}$ Low	10		μs

EPROM PROGRAMMING AND VERIFICATION WAVEFORMS

WF025692

For Programming conditions, see Figures 1 and 2.
For Verification conditions, see Figure 3.

87C521/87C541

Software Routines

DUAL DATA POINTER ROUTINES

The Dual Data Pointer feature enhances the manipulation of external memory by providing an easy way to use two separate 16-bit pointers with external memory and to selectively switch between them. This can increase execution speed of many functions considerably while at the same time reducing the number of required instructions. For instance, in block-move operations in external RAM, Dual Data Pointers can show more than 100% speed improvement using less than 65% of the original code space.

The following registers are associated with the Dual Data Pointers.

Data Pointer Low   (DPL) ⎫

Data Pointer High   (DPH) ⎬ DPTR0 (Original Data Pointer)

Data Pointer Low 1   (DPL1) ⎫

Data Pointer High 1   (DPH1) ⎬ DPTR1 (New Data Pointer)

Data Pointer Selection   (DPS)

The six instructions that refer to "DPTR" now refer to the data pointer that is currently enabled, either DPTR0 or DPTR1. DPS is used to selectively enable the data pointers.

```
INC     DPTR            ; Increment Data Pointer
MOV     DPTR,#data16    ; Loads DPTR with 16-bit constant
MOVC    A,@A+DPTR       ; Move code byte relative to DPTR to Acc
MOVX    A,@DPTR         ; Move external RAM to Acc
MOVX    @DPTR,A         ; Move Acc to external RAM
JMP     @A + DPTR       ; Jump indirect relative to DPTR
```

For complete information on the Dual Data Pointer feature, consult the 80C521/80C321 Data Sheet.

Block Move in External RAM

Data Pointers are used extensively in the 8051 Family when a block of data is moved from a source area to a destination area in external RAM. The following examples illustrate the speed improvement and code space efficiency gained by using the Dual Data Pointer feature.

The first example shows a 32-byte block move executed by a traditional, single data pointer 8051 Family member. Contrast this with the second example which shows a 32-byte block move executed using the Dual Data Pointers.

With Dual Data Pointers, one data pointer can be assigned to the source address and the other to the destination address. The code then switches between the two data pointers without having to save and restore a data pointer. The speed improvement of this 32-byte block move is 115% and uses less than 57% of the original code space.

32-Byte Block Move with a Single Data Pointer

```
; SH and SL are the High and Low source addresses
; DH and DL are the High and Low destination addresses
; Register R5 contains the number of bytes to be moved

;                              Bytes/Cycles
;
        MOV   R5,#32       ;  2  1  - 32 bytes to move
        MOV   DPTR,#SHSL   ;  3  2  - Source      address
        MOV   R1,#SL       ;  2  1  - Initialize source address
        MOV   R2,#SH       ;  2  1
        MOV   R3,#DL       ;  2  1  - Initialize dest. address
        MOV   R4,#DH       ;  2  1

LOOP:   MOVX  A,@DPTR      ;  1  2  - Read byte from source
        MOV   R1,DPL       ;  2  2  - Save source pointer
        MOV   R2,DPH       ;  2  2
        MOV   DPL,R3       ;  2  2  - Load dest. pointer
        MOV   DPH,R4       ;  2  2
        MOVX  @DPTR,A      ;  1  2  - Write byte to dest.
        INC   DPTR         ;  1  2  - Next dest. pointer
        MOV   R3,DPL       ;  2  2  - Save dest. pointer
        MOV   R4,DPH       ;  2  2
        MOV   DPL,R1       ;  2  2  - Load source pointer
        MOV   DPH,R2       ;  2  2
        INC   DPTR         ;  1  2  - Next source pointer
        DJNZ  R5,LOOP      ;  2  2  - Loop till R5=0
```

32-Byte Block Move with Dual Data Pointers

```
; SH and SL are the High and Low Source addresses
; DH and DL are the High and Low Destination addresses
; Register R5 contains the number of bytes to move
; DPS = 01 at start (DPTR1 selected)

;                              Bytes/Cycles
;
        MOV   R5,#32       ;  2  1  - 32 bytes to move
        MOV   DPTR,#DHDL   ;  3  2  - DPTR1 = Dest. address
        INC   DPS          ;  2  1  - Switch to DPTR0
        MOV   DPTR,#SHSL   ;  3  2  - DPTR0 = Source address

LOOP:   MOVX  A,@DPTR      ;  1  2  - Read byte from source
        INC   DPS          ;  2  1  - Switch to DPTR1
        MOVX  @DPTR,A      ;  1  2  - Write byte to dest.
        INC   DPTR         ;  1  2  - Next dest. pointer
        INC   DPS          ;  2  1  - Switch to DPTR0
        INC   DPTR         ;  1  2  - Next source pointer
        DJNZ  R5,LOOP      ;  2  2  - Loop till R5=0
```

Suggestion: The fastest way to switch data pointers is to increment the DPS register. Since Bits 7–1 of this register are defined to be zero, the increment (or decrement) operation simply alternates the contents of DPS between 00H and 01H.

32–Byte Block Move Efficiency		
	Single Data Pointer	Dual Data Pointers
Instructions	19	11
Bytes	35	20
Cycles	839	390
Time (μs) @16 MHz	629.25	292.5

N–Byte Block Move Efficiency (Where N < 256)		
	Single Data Pointer	Dual Data Pointers
Instructions	19	11
Bytes	35	20
Cycles	26N + 6	12N + 6
Time (μs) @16 MHz	0.75 (Cycles)	0.75 (Cycles)

Higher Performance Interrupt Routines

When a frequently occurring interrupt uses a data pointer, the overhead required to store and reload it from the main program can be significant. The performance of interrupt-driven systems can be improved by using the Dual Data Pointer feature to assign a data pointer to a frequently called, time-critical interrupt routine.

In the following code, the Main routine uses only DPTR0. The Interrupt routine stores a byte from the Serial Port into an external RAM buffer for later processing. DPTR1 is dedicated for its use.

```
RESET:      SJMP    START


START:      MOV     DPTR,#MAIN      ; Main routine data pointer
            INC     DPS             ; Switch to DPTR1
            MOV     DPTR,#INT       ; Interrupt data pointer
                                    ; initialization
            INC     DPS             ; Switch back to DPTR0
            MOV     IE,#90H         ; Enable Serial Port Int.

; Main routine is using DPTR0
;           ...     .....
;           ...     .....
;                   ----->>> Interrupt occurs
; Program continue
;           ...     .....
;           ...     .....
;           ...     .....
;           ...     .....

; Interrupt routine begins at the Serial Port Vector Address

VECTOR:     INC     DPS             ; Switch to DPTR1
            MOV     A,SBUF          ; Read from Serial Port
            MOVX    @DPTR,A         ; Store byte in RAM Buffer
            INC     DPTR            ; Next Dest. Address
            INC     DPS             ; Switch to DPTR0
            RETI                    ; Return from Interrupt
```

Full Duplex Transmit/Receive Buffering

Full Duplex Serial Port operation involves simultaneously transmitting and receiving data. Typically a separate transmit buffer and a receive buffer are assigned in the external memory. When a receive interrupt occurs, the data received in the serial port receive register is saved in the external receive buffer. When data is ready to be transmitted, the data from the external transmit buffer is loaded into the transmit register of the serial port. With two data pointers available, one can be assigned to the transmit buffer and the other to the receive buffer. Thus, the interrupt overhead can be reduced.

```
; Initialize
            MOV     DPS,#00H        ; Select DPTR0
            MOV     DPTR,#XMTBUF    ; Transmit RAM buffer address
            INC     DPS             ; Switch Data Pointers
            MOV     DPTR,#RCVBUF    ; Receive RAM buffer address

; Serial Port Interrupt Routine

INT_BEGIN:  JB      RI,RECEIVE      ; Receive a Byte
            JB      TI,TRANSMIT     ; Transmit a Byte
            SJMP    ERROR           ; Error - neither bit set

TRANSMIT:   CLR     TI              ; Clear Flag
            MOV     DPS,#00H        ; Select DPTR0
            MOVX    A,@DPTR         ; Load data from memory
            MOV     C,P             ; Move Parity bit to carry bit
            CPL     C               ; Set ODD Parity
            MOV     A.7,C           ; Append to bit 7 in Acc
            MOV     SBUF,A          ; Load data to transmit
            INC     DPTR            ; Next Byte
            RETI

RECEIVE:    CLR     RI              ; Clear Flag
            MOV     A,SBUF          ; Load received byte to Acc
            JNB     P,ERROR         ; Jump if Parity error
            ANL     A,#7FH          ; Mask off Parity bit
            MOV     DPS,#01H        ; Select DPTR1
            MOVX    @DPTR,A         ; Store byte in memory
            INC     DPTR            ; Next byte
            RETI

ERROR:      ...     .....           ; Error Handler
            RETI
```

Tree Structure Manipulation

The Dual Data Pointers can be useful in applications involving data structures containing pointer references, such as trees. For instance in a tree search algorithm, the node currently being searched and its parent may have their addresses stored in the Dual Data Pointers. Even though other required pointers will necessarily be pushed onto the stack, most operations will involve only the two most recently used data pointers. Thus the search algorithm will execute more quickly.

In Figure 8-1, note that DPTR1 can be used to step through another link at node "Dave", as soon as DPTR0 is through accessing all of the links in leaf-node "Jim". The pointer for node "Randy" is located on the stack at this point.
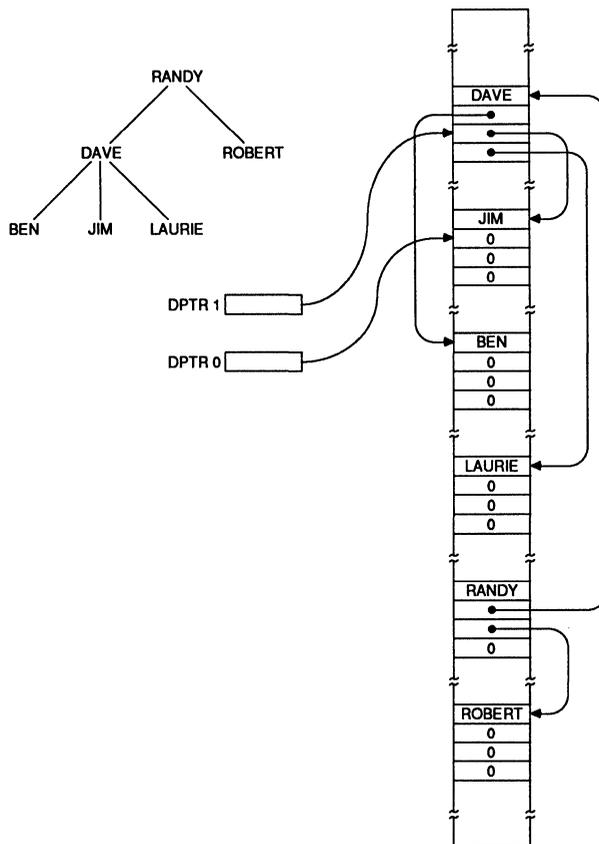
09757A

Figure 8-1. Tree Structure in External Memory

ROM Table Access

Use of the Dual Data Pointers need not be limited to manipulations in external RAM. For instance, one or both data pointers can be assigned to ROM tables in program memory space. Table access is then performed with the MOVC instruction. In this way, the base address of a ROM table can reside in one of the data pointers, improving the effective access time.

Creating an External Stack

For applications that require large amounts of data to be stored on a stack, the internal RAM space may not be sufficient to contain it. This is especially true if the internal RAM is already being used extensively.

With Dual Data Pointers, one data pointer can be assigned specifically to an external stack space in external RAM. The following code provides Push and Pop subroutines using DPTR1 as a stack pointer. Two examples are shown. In the first example the external stack may be up to 64K bytes in length. The second example executes more quickly, but the external stack is limited to 256 bytes.

Example 1 — 64K byte External Stack Space

```
; Both Routines Push/Pop bytes from/to the Accumulator


PUSH:       INC     DPS             ; Switch to DPTR1
            INC     DPTR            ; Increment DPTR1
            MOV     @DPTR,A         ; Move Accumulator to Stack
            INC     DPS             ; Switch back to DPTR0
            RET

POP:        INC     DPS             ; Switch to DPTR1
            MOV     A,@DPTR         ; Move Stack byte to Acc
            CJNE    DPL1,#00H,LOW   ;
            DEC     DPH1            ;

LOW:        DEC     DPL1            ; Decrement DPTR1
            INC     DPS             ; Switch back to DPTR0
            RET
```

Example 2 — 256 Byte External Stack Space

```
PUSH:       INC     DPS             ; Switch to DPTR1
            INC     DPL1            ; Increment DPTR1
            MOV     @DPTR,A         ; Move Accumulator to Stack
            INC     DPS             ; Switch back to DPTR0
            RET

POP:        INC     DPS             ; Switch to DPTR1
            MOV     A,@DPTR         ; Move Stack byte to Acc
            DEC     DPL1            ; Decrement DPTR1
            INC     DPS             ; Switch back to DPTR0
            RET
```

WATCHDOG TIMER ROUTINES

The Watchdog Timer (WDT) is a specially designed timer that will reset the chip upon reaching a pre-programmed time interval. Once started it cannot be disabled, except by a reset. It allows safe recovery from problems resulting from electrostatic discharge, external noise, unexpected input conditions or external events, and programming anomalies. Two registers are associated with the Watchdog Timer:

Watchdog Selection (WDS)
Watchdog Key (WDK)

WDS is used to set up the programmed time intervals and indicates the cause of the last reset — a Watchdog or Software Reset versus a Hardware or Power-on Reset. Sixteen time intervals are programmable varying from 128 µs to 4 s (at 12 MHz).

WDK is used to enable the Watchdog Timer as well as clear it. When the Watchdog Timer is cleared, its present count is set to zero, but it continues to increment. For complete information on the Watchdog Timer, consult the *80C521/80C321 Data Sheet.*

WDT Enable, Clear, and Reset Cause

The following example shows a method of setting up the Watchdog time value to 16.384 ms assuming a 12 MHz clock. The Watchdog Timer is then enabled.

```
; Enable Watchdog Timer
            MOV     WDS,#07H        ; Set up 16.384 msec
                                    ;
            MOV     WDK,#A5H        ; Write first key value
            MOV     WDK,#5AH        ; Write second key value
                                    ; Watchdog timer is 'enabled'
```

Once the Watchdog Timer is enabled, a "clear" sequence should be performed at intervals not exceeding the 16.384 ms time value. The enabling sequence may be used to clear the Watchdog Timer.

```
; Clear Watchdog Timer
        MOV     WDK,#A5H        ; Write first key value
        MOV     WDK,#5AH        ; Write second key value
                                ; Watchdog Timer is 'cleared'
                                ; but continues to increment.
```

To test whether the last reset was caused by a Watchdog or Software Reset the following code may be used. If the Reset Cause bit is set, then a Watchdog or Software Reset has occurred.

```
; Reset Cause Identification
        MOV     A,WDS           ; Read Watchdog Selection reg.
        JB      A.7,WDRST       ; Jump if Reset Cause bit is
                                ; set, else continue
WDRST:  ...     .....           ; Notify external circuitry
```

The security of the Watchdog Timer is not adversely affected by interrupts that may occur in between the writing of the 'A5' and '5A' values to the WDK Register. Thus, if necessary, the user may include clear operations within both a main routine and the interrupt routines. Furthermore, the user need not disable interrupts during the enable/clear operations.

Once the 'A5' is written to WDK, the interrupt routine can only affect the Watchdog Timer in three ways: 1) it can go ahead and enable/clear the Watchdog Timer with a '5A'. (The subsequent '5A' written by the main routine will then have no effect); 2) it can write another 'A5'. This affects neither the Watchdog Timer nor the main routine; or 3) it can cause a Software Reset by writing a value other than 'A5' or '5A'. Any routine, though, can be written to generate the Software Reset.

Power-Down Operation

While the Watchdog Timer is enabled, the Power-Down mode is disabled. The user's code may still attempt to enable the power-down operation (by writing a value 1 to the PD bit in the PCON register), however, the PD bit will remain at 0, and the power-down operation will not take place. If the WDT has *not* been enabled, the power-down operation can proceed normally.

To enter Power-Down mode when the WDT is enabled, the WDT must first be disabled via a Hardware Reset, Software Reset, or Watchdog Reset. The easiest is the Software Reset. This can be accomplished by writing an 'A5' to the Watchdog Key (WDK) register followed by a value other than 'A5' or '5A'. This generates an immediate reset, equivalent to a Hardware Reset except that the Reset-Cause bit is set.

The code below uses the Reset-Cause bit and the Internal RAM (which is not modified by a reset). If the Reset-Cause bit is set, and a special Power-Down-Status byte in internal RAM contains '88H', then the Power-Down mode will be entered by the program code.

```
; WDS = 7 sets up a Watchdog time of 16.384 msec @ 12 MHz.
; 'A5' followed by '5A' written to WDK enables the WDT.
; RAM location 50H is Power Down Status
;       00 implies Power-Down has not been requested.
;       88 implies Power-Down has been requested.

RESET:      MOV     A,WDS           ; Read Reset cause bit in WDS
            JB      A.7,WDRST       ; Jump if reset caused by WDT
            LJMP    MAIN            ; Go on to the Main Routine

WDRST:      MOV     R0,#50H         ; Address Power Down Status
            CJNE    @R0,#88H,MAIN   ; If Power-Down was not
                                    ; requested, then jump and
                                    ; continue normally
            MOV     PCON,#02H       ; else enter Power-Down Mode

MAIN:       MOV     50H,#00H        ; Clear Power Down Status
            MOV     WDS,#07H        ; Set up time value for WDT
            MOV     WDK,#A5H        ; Write first key value
            MOV     WDK,#5AH        ; Write second key value
;           ...     .....           ; WDT is now enabled.
;
; Main Routine Continues..
;
; In Main Routine whenever Power-Down is required, execute:

            MOV     50H,#88H        ; Request Power Down operation
            MOV     WDK,#A5H        ; Write first key value
            MOV     WDK,#11H        ; Software Reset generated -
            NOP                     ; Execution begins at RESET
                                    ; in 3 machine cycles.
```

Testing the Watchdog Timer

Two methods can be used to verify that the WDT is enabled after the enabling sequence has been written (rather than simply waiting for the WDT to reset to occur). Method I can be used as a precautionary measure after the enabling sequence or at various points within the code. It may also be used to confirm the time interval programmed into the WDT for applications that occasionally use different Watchdog time intervals. Method II can be used as a debugging test during program development.

Method I

```
          MOV     WDS,#07H        ; Set the Watchdog time to
                                  ; 16.384 ms @12 MHz
          MOV     WDK,#A5H        ; Write first key value
          MOV     WDK,#5AH        ; Write second key value
                                  ; WDT should now be enabled
          MOV     WDS,#00H        ; Attempt to rewrite contents
                                  ; of the WDS Programmed Time
          MOV     A,WDS           ; Read contents of WDS into Acc
          CJNE    A,#07,ERROR     ; If contents are not 07, then
                                  ; jump to ERROR.
          ...     .....           ; The WDT is enabled and the
                                  ; ACC now holds the programmed
                                  ; time value that the WDT is
                                  ; currently using.

ERROR:    ...     .....           ; Watchdog Timer never received
                                  ; the correct 'A5-5A' sequence
```

Method II

```
          MOV     WDS,#07H        ; Set the Watchdog time to
                                  ; 16.384 ms @12 MHz
          MOV     WDK,#A5H        ; Write first key value
          MOV     WDK,#5AH        ; Write second key value
                                  ; WDT should now be enabled

WAIT:     MOV     A,WDS           ;
          JNB     A.5,WAIT        ; Wait 4.096 ms for the TV bit
                                  ; to be set
          ...     .....           ; WDT enabled and incrementing
```

Using the Watchdog Timer as a Standard Timer

The Timer Verification (TV) bit in the WDS register can be used to implement certain types of timer functions through polling. Once the WDT is enabled, the TV bit will toggle every 4.096 ms (at 12 MHz) until either the WDT overflows, or the WDT is cleared. (The TV bit is initially a 0 after any reset.) When the WDT overflows, a WDT Reset occurs clearing the TV bit. When the WDT is cleared, the TV bit is cleared, but begins toggling again at the same rate. If bits PT3–PT0 are set to '0101' or less, then a WDT Reset will occur before the TV bit toggles.

The following code uses the MAIN polling loop of an application to watch for the TV bit to toggle. It uses the TV bit to output a 25% duty-cycle pulse on Port Pin 1.7 with a period of 1.049 s at 12 MHz.

```
; R6    If 0, then Pulse is Low
;       If 1, then Pulse is High
; LTIME = Low Time, the number of 4.096 ms units equaling
;         786 ms = 192
; HTIME = High Time, the number of 4.096 ms units equaling
;         262 ms = 64
; OLD_TV = A Direct RAM byte whose bit 0 location contains
;          the last read value of TV
; R7 Contains number of TV toggles left to go before P1.7
;    switches

INIT:       MOV     WDS,#0FH        ; Set the Watchdog time to 4 S
                                    ; at 12 MHz (safest value)
            CLR     P1.7            ; Set Port Pin to 0
            MOV     R6,00H          ; Pulse is Low
            MOV     R7,LTIME        ; Load Low Time
            MOV     WDK,#A5H        ;
            MOV     WDK,#5AH        ; WDT is now enabled. TV begins
                                    ; toggling
            MOV     R6,00H          ; Pulse is Low
            MOV     R7,LTIME        ; Load Low Time
            MOV     OLD_TV,#00H     ; Old TV bit equals 0 (TV's
                                    ; reset value)

MAIN:       ...     ....
            MOV     A,WDS           ;
            MOV     C,A.5           ; Move TV bit to Carry
            MOV     A,OLD_TV        ; Move Old TV bit to ACC.0
            ADDC    A,#00           ; Add TV bit (in Carry) to Old
                                    ; TV bit
            JB      A.0,TOGGLE      ; If A.0 = 1, then the TV bit
                                    ; has toggled

CONTINUE:   ...     ....
;           ...     ....

TOGGLE:     INC     OLD_TV          ; Toggle Old TV bit in OLD_TV
                                    ; byte
            DJNZ    R7,CONTINUE     ; If R7 is not 0, then it is
                                    ; not time to toggle P1.7 yet
            CPL     P1.7            ; Toggle Port Pin
            MOV     WDK,#A5H        ;
            MOV     WDK,#5AH        ; Clear WDT, TV starts again
            CJNE    R6,#00,GO_LOW   ; If R6 is 0, then load HTIME
                                    ; else load LTIME
            MOV     R7,HTIME        ; Load High Time
            INC     R6              ; Pulse is High now
            SJMP    CONTINUE        ;

GO_LOW:     MOV     R7,LTIME        ; Load Low Time
            DEC     R6              ; Pulse is Low now
            SJMP    CONTINUE        ;
```
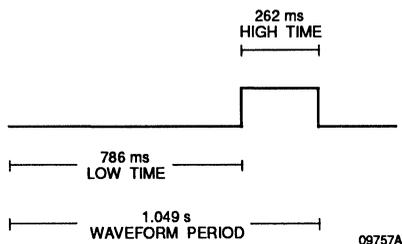
Figure 8-2. P1.7 Output — 25% Duty Cycle

SOFTWARE RESET ROUTINES

A Software Reset may be accomplished through the Watchdog Timer. This "software generated" Watchdog Reset occurs regardless of whether or not the Watchdog Timer was previously enabled. If the Watchdog Timer was enabled, it will be disabled following the reset. The Software Reset is functionally equivalent to the Watchdog Reset.

Two write operations are required to initiate a Software Reset to greatly reduce the chance of unintentional Software Reset generation. More information is available in the *80C521/80C321 Data Sheet*.

Using Software Reset

Whether or not the Watchdog Timer is being used, the Software Reset feature of the Watchdog Timer may be used to increase the reliability of the program code. For instance, the detection of an unusual hardware error can be followed by a jump to the following code which will always cause a Software Reset.

```
CLR     EA              ; Disable all interrupts.
                        ; Optional
MOV     FLAG,#88H       ; Optional
MOV     WDK,#A5H        ; Write first key value
MOV     WDK,#11H        ; Write a non-A5, non-5A value.
                        ; Software Reset has now been
                        ; generated via the WDT.
NOP                     ; Optional
```

If the Watchdog Timer is cleared within an interrupt routine, that interrupt should be disabled before executing a Software Reset sequence. If the interrupt occurs between the two writes to WDK, and then clears the Watchdog Timer, a Software Reset will not be generated.

To distinguish between a Watchdog Reset and a Software Reset (or separate causes of a Software Reset), a flag value may be written to internal RAM. This flag can be used in combination with the Reset-Cause bit to distinguish between the reset types. An example of this method is shown in the "Power Down Operation" software routine.

After the value '11H' is written to WDK, execution begins at 0000H in three machine cycles. One machine cycle of normal execution takes place after the '11H' is written. Thus, the NOP can be included for safety. Since all registers are initialized during reset, and all external operations take two machine cycles, the only operation that could possibly affect operation after the Software Reset would be a one-cycle write to internal RAM.

Improving Reliability with Software Reset

For additional reliability, the following instruction sequence may be placed in any unused ROM program space:

```
            NOP           ; First unused ROM location
            NOP
;
            MOV     WDK,#A5H
            MOV     WDK,#00H    ; Software Reset generated
            NOP
            NOP
;
            MOV     WDK,#A5H
            MOV     WDK,#00H    ; Software Reset generated
            NOP
            NOP
;
;           ...     ....        ; Continue repeating the 4-instruction
;           ...     ....        ; sequence
;

SOFTRESET:  MOV     WDK,#A5H
            MOV     WDK,#00H    ; Software Reset generated
            NOP
            NOP
            SJMP    SOFTRESET   ; Last unused ROM location
```

If the program counter branches to any byte of this code (other than the second byte of the SJMP instruction), a Software Reset will be quickly generated. The NOP instructions are used to force the program counter to adjust itself to an instruction boundary.

CHAPTER 9

80C324 CMOS Single-Chip Microcontroller

80C324 Data Sheet   **9-1**

80C324
CMOS Single-Chip Microcontroller

**Advanced
Micro
Devices**

DISTINCTIVE CHARACTERISTICS

- **Software and pin-compatible with 80C321 and industry standard 80C31**

- **Port Expansion Mode added to 80C321**
  - Capability for up to 15 8-bit I/O ports
  - Software identical to on-chip I/O ports
  - Simple external hardware construction
    - Multiplexed through Port 1
    - EA/PXS pin used for strobe timings

- **All 80C321 features retained**
  - 256 bytes RAM
  - Dedicated Watchdog Timer
    - Robust: Immune to software disables
    - Flexible: User programmable from 128 μs to 4 seconds @ 12 MHz
  - Dual Data Pointers
    - Faster external memory access
  - Software Reset

GENERAL DESCRIPTION

The 80C324 is a superset of the 80C321 and industry standard 80C31 architectures. The 80C324 provides an expansion capability for adding additional external I/O ports to the microcontroller. In this mode, up to 15 high-performance I/O ports can be utilized by the 80C324 CPU without sacrificing any on-chip functionality.

This product retains all of the features of a 80C321, including a programmable Watchdog Timer and Dual Data Pointers to enhance reliability and improve performance. For information on these features see the 80C321 data sheet.

PORT EXPANSION MODE

Port Expansion Mode (PEM) provides capability for up to 15 full speed I/O ports. Fourteen additional I/O ports can be constructed externally by multiplexing through Port 1 and using EA/PXS for strobe timing. Port 3 operates as normal; however, all other ports, including Port 0 and Port 2, which normally are sacrificed for a multiplexed data/address bus, are reconstructed.

The new ports are accessed by software exactly as if they existed on-chip. The entire 8051 instruction set is available for these additional ports. Traditional memory-mapped I/O ports allow only four instructions to be used, vastly reducing their effectiveness.

Although the address decode logic, latches, and buffers exist externally, the 80C324 behaves as if these ports were mapped into the internal SFR (Special Function Register) space. The SFR address locations for the expanded external ports are shown in Table 1.

When Port Expansion Mode is enabled, Port 1 pins become the Port Expansion Bus, which contains the information necessary to build ports externally. Port 1 may not be used as a standard port in Port Expansion Mode; however, it may be rebuilt externally, if desired. Port 3 always exists on-chip and is not affected by Port Expansion Mode. The total number of possible ports, including those on- and off-chip, is 15.

Table 1. Reserved Set of SFR Addresses in PEM

Address:	Name:
*90H	Port 1 (P1)
91H	
92H	
93H	
94H	
95H	
96H	
97H	
*C0H	Port 7 (P7)
*D8H	Port 6 (P6)
*E8H	Port 4 (P4)
*F8H	Port 5 (P5)
*80H	Port 0 (P0)
*A0H	Port 2 (P2)
‾‾14‾‾	
	(Port 3 on-chip)
‾‾15‾‾	Total

Port Expansion mode feature is only available in 8031 mode (that is, $\overline{EA}$ is Low)

*Bit-Addressable Port (only Bit-Addressable Ports are given formal names)

The instructions that can operate on the external ports during Port Expansion Mode include all instructions that access direct addresses or bit addresses. Table 2 shows these instructions and the type of access that is performed on the direct or bit address—Read-only, Write-only, or Read/Write. Consult the 8051 Family Instruction Set for full details. The MOVX instructions (and therefore the Dual Data Pointers) are no longer needed to access external ports, saving both time and code space.

Table 2. Instructions Referencing Direct or Bit Addresses

Direct Information		Type of Access	Bit Instruction		Type of Access
ADD	A,direct	Read	CLR	bit	Read/Write
ADDC	A,direct	Read	SETB	bit	Read/Write
SUBB	A,direct	Read	CPL	bit	Read/Write
INC	direct	Read/Write	ANL	C,bit	Read
DEC	direct	Read/Write	ANL	C,/bit	Read
ANL	A,direct	Read	ORL	C,bit	Read
ANL	direct,A	Read/Write	ORL	C,/bit	Read
ANL	direct,#data	Read/Write	MOV	C,bit	Read
ORL	A,direct	Read	MOV	bit,C	Read.Write
ORL	direct,A	Read/Write	JB	bit,rel	Read
ORL	direct,#data	Read/Write	JNB	bit,rel	Read
XRL	A,direct	Read	*JBC	bit,rel	Read/Write
XRL	direct,A	Read/Write			
XRL	direct,#data	Read/Write			
MOV	A,direct	Read			
MOV	Rn,direct	Read			
MOV	direct,A	Write			
MOV	direct,Rn	Write			
+MOV	direct,direct	Read/Write			
MOV	direct,@Ri	Write			
MOV	direct,#data	Write			
MOV	@Ri,direct	Read			
PUSH	direct	Read			
POP	direct	Write			
XCH	A,direct	Read/Write			
CJNE	A,direct,rel	Read			
DJNZ	direct,rel	Read/Write			

*This instruction reads the bit twice.
+This instruction normally reads from one address and writes to another.

Enabling/Disabling Port Expansion Mode

If Port Expansion Mode (PEM) is not enabled, the architecture and operation of the 80C324 ports is identical to that of the 80C31. Port Expansion Mode can be enabled by either hardware or software. These two options are termed Hardwired PEM and Software PEM. They offer different methods of entering/existing PEM, but behave identically in every other respect.

Hardwired PEM

Hardwired PEM is enabled by placing 80C324 pins into specific states before the falling edge of the reset pulse on the RST pin. These values are latched internally on the falling edge of the reset pulse during a Hardware or Power-on Reset. After the reset pulse the port pins should be driven to their initialization values by the user. The drivers for ALE and $\overline{\text{PSEN}}$ should be three-stated by the user at the falling edge of reset.

Hardwired PEM Enabling Requirements:

	@ Falling RST	@ Falling RST + 450 ns
*ALE	Low	High
*$\overline{\text{PSEN}}$	Low	High
**P2.7	High	Don't Care
P2.6	Low	Don't Care

*ALE and $\overline{\text{PSEN}}$ have weak internal pullups that will pull these pins High within 450 ns if they are not externally driven Low by the user.

** During a reset sequence, this pin will be pulled High internally and remain High, unless externally driven Low by the user.

Once Hardwired PEM is enabled, it operates uninterrupted until a Hardware Reset (the Software and Watchdog Reset types do not disable Hardwired PEM). During Hardware Reset, the defined values must again be present on ALE, $\overline{\text{PSEN}}$, P2.7, and P2.6 at the falling edge of the pulse on the RST pin, or Hardwired PEM will be disabled after reset. Hardwired PEM has priority over Software PEM.

Software PEM

Software PEM is enabled through the Output Function Enable (OFE) register. This register is Read/Write. If Hardwired PEM is enabled, modifying this register will not affect Port Expansion Mode. Its default value after any reset is 00H.

Output Function Enable—(OFE)
Address: AC (Hex)

(MSB)							(LSB)
0	0	0	0	0	PEME	0	0
7	6	5	4	3	2	1	0

Bits 1–0
Reserved. Will return 0 when read.

Bit 2—Port Expansion Mode Enable (PEME)
If this bit is set to a 1, Software PEM will be enabled. If this bit is 0, Software PEM will be disabled. The default value of PEME after any reset is 0 (the state of this bit may be modified by the user during Hardwired PEM, but it will not affect any chip operation). Since OFE is not bit-addressable, this bit must be set with a direct instruction.

Bits 7–3
Reserved. Will return 0 when read.

Port Expansion Bus

Port 1 pins on the 80C324 are used to carry information required to build ports external to the device. An additional control signal, Port Expansion Strobe (PXS), is provided that decreases the amount of external port expansion logic required. PXS is an output of the $\overline{\text{EA}}$ pin. (The $\overline{\text{EA}}$ value is latched at every Hardware or Power-on Reset, freeing this pin for the PXS function.)

When PEM is enabled, Port 1 changes state on every Phi 2 transition of the clock, except when three-stated (Figure 1). PXS and ALE decode various strobes that are needed by the external logic, as shown in Table 3.

Table 3. Port Expansion Bus Control Logic

ALE	PXS	Function	Description
High	Low to High	Address Strobe	External Port Address should be latched from Port 1
High	High to Low	Read Strobe	Read Data from the most recently latched address should be driven onto Port 1
Low	Low to High	Write Strobe	Write Data should be latched from Port 1 into the most recently latched external port address
Low	High to Low	No Information	Switches state of PXS without affecting PEM
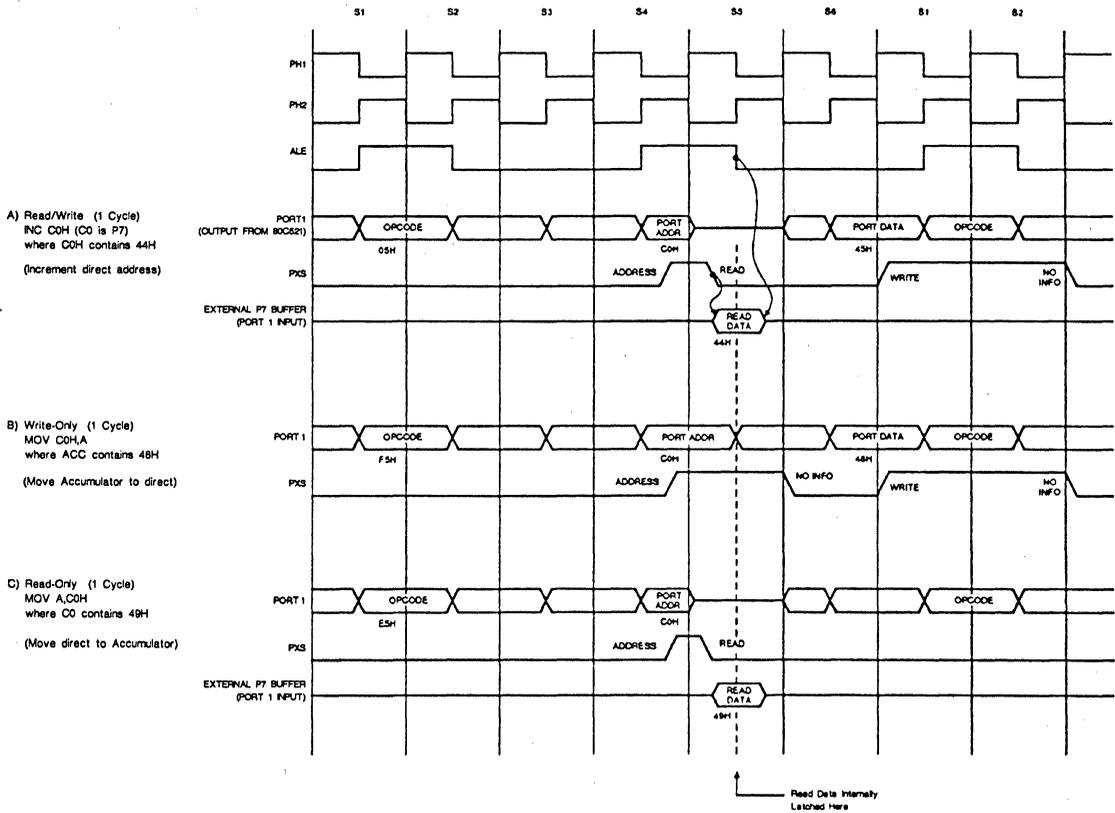
Figure 1. Three Examples of Port Expansion Bus Operation

WF025511

80C324

It is convenient to describe the operation of the Port Expansion Bus from the standpoint of the various 8051 instructions that may be executed by the user on the 80C324.

If the instruction being executed does not reference a direct or bit address, Port 1 may switch, but PXS will stay Low (that is, inactive).

If the instruction references a direct or bit address within the Reserved Set of PEM addresses (see Table 1), Port 1 and PXS will switch as shown in Figure 1. In examples 1a and 1c, Port 1 is three-stated during S5 in order to allow the read data to be driven back onto the Port Expansion Bus. Read data will be internally latched by the 80C324 from the Port Expansion Bus at the beginning of S5P2 (State 5 Phi 2). In example 1b, read data is not required.

If the instruction references a direct or bit address that is not within the Reserved Set, both PXS and Port 1 will still switch as shown in Figure 1, except that Port 1 will *never* be three-stated during S5. Thus, it is required that the user fully decode the address of each external port to avoid contention on the Port Expansion Bus when Read Data is required (Note: There are certain configurations where only the upper 5 bits of the address are required. See Tier 1.)

If a bit instruction is executed, the bit address, rather than the port address, will be provided on Port 1. When a bit address is provided, the entire port byte to which it belongs must be supplied as Read Data. The port address to be supplied is easily decoded from the bit address, as it is simply the upper 5 bits of the direct address (e.g., bit address FBH implies port address F8H). The Write Data provided by the bit instruction forms the entire byte that should be written to the port.

Table 4 gives examples of the preceding cases. The "SJMP label" instruction does not involve a direct address. The "INC direct" instruction performs a read of the direct address, followed by a write of the incremented value. The "SETB bit" instruction reads the direct byte associated with the bit address, sets the bit location within that direct byte, and rewrites the modified direct byte.

For Hardwired PEM, the Port Expansion Bus operates continuously, except during reset. For Software PEM the Port Expansion Bus operates starting at S3P1 of the instruction following the one that set the PEME bit in the OFE register. If Software PEM is turned off, the Port Expansion Bus goes back to normal behavior, starting at S3P1 of the instruction following the one that cleared the PEME bit.

Entry into Idle Mode is possible during PEM. Both Port 1 and PXS will be internally pulled High during Idle. PEM will remain enabled if Idle Mode is exited via an interrupt.

Entry into Power-Down Mode is possible during PEM. Both Port 1 and PXS will be internally pulled High during Power-Down. The Hardware Reset that follows will disable both Hardwired and Software PEM; thus PEM must be re-enabled after a Power-Down, if desired.

External Logic Implementation

Two tiers of implementation are possible. With just PXS and address decode logic, most of the capabilities of on-chip ports can be provided externally. These capabilities are described in Tier 1. To exploit every possible capability, however, additional external logic must be provided to decode the opcodes of instructions as they are executing on the 80C324. These capabilities are described in Tier 2. The two tiers differ in external logic implementation only; the 80C324 operates identically in both cases. Tier 1 is a lower-cost solution and is probably sufficient for most applications.

Table 4. Examples of Port Expansion Bus Operation

Inst	Description	Port 1 and PXS	Comments
SJMP label	Short Jump to label	PXS stays Low, Port 1 switches	Does not reference a direct address
INC C0H	Increment direct	Operates as in Figure 1	The direct address for Port 7 is within the reserved set of PEM addresses
INC C3H	Increment direct	PXS switches, Port 1 switches	Since direct address C3H is not in the reserved set, Port 1 is not three-stated during S5. PXS still switches as in Figure 1
SETB C3H	Set direct bit	Same as Figure 1 except the Port Address is C3H	Since bit address C3H is bit 3 of Port 7 (C0H), Port 1 is three-stated during S5.

The Simple Approach—Tier 1

Tier 1 consists of using Port 1 pins, Port Expansion Strobe (PXS), and ALE. In Tier 1 the following capabilities are possible.

The user may build up to 14 External Ports, with 5 being Bit Addressable; or up to 6 External Ports, with all 6 being Bit Addressable. As shown in Table 1, Port locations A0, C0, D8, E8, and F8H are Bit Addressable. If a sixth Bit Addressable port is needed, Port 1 (address 90H) can be made Bit Addressable if addresses 91–97H are not implemented as external port addresses by the user and never referenced as such, but reserved by the user as the bit addresses of Port 1.

Ports A0, C0, D8, E8, and F8H each have the next 7 sequential addresses reserved in the 80C324, and are therefore Bit Addressable in Tier 1 (e.g., if address DAH is latched, the external logic can assume that it is referring to bit 2 of Port D8H, since no use for direct address D8H exists on-chip).

All instructions can reference the new external ports except "JBC bit,label." Unlike other 8051 instructions, this instruction is implemented with two consecutive "read" operations. The PXS, ALE combination is not sufficient to decode this case. The "JBC bit,label" instruction, however, will still work as defined for any on-chip ports while PEM is enabled. (Also, the JBC instruction can be replaced with a "JB bit,label" followed by a "SETB bit" instruction at the branch address. If the bit was already cleared, no time is added. If the bit was set, one machine cycle is added.)

Read/Modify/Write (RMW) instructions are a subset of the Read/Write instructions listed in Table 2. For the on-chip ports, the RMW instructions read the port LATCH (output) rather than the Port PINS (input) (i.e., they read what was last written rather than what is currently present on the input pins). No such distinction can be implemented with external ports in Tier 1, since PXS furnishes only one type of Read Data strobe. The RMW instructions, of course, may be freely used on the external ports; however, the user should be aware that they may not operate identically to the on-chip ports in all designs.

For example, if an on-chip port pin is directly driving the base of a transistor, the internal latch may contain a logical 1, while the actual voltage level on the pin is only 0.7 V. Thus, a RMW instruction would supply the value 1, while an instruction that reads the pins would supply a logical 0. For an external port, the Read Data strobe is used to enable the external read buffer, always resulting in a logical 0 being sent to the 80C324. For most applications, however, this distinction in operation will not be seen.

The RMW instructions will still work in the customary manner for any on-chip ports, whether or not PEM is enabled. All Read/Write instructions are shown in Table 5.

Table 5. Read/Write Instructions

Read/Modify/Write Instructions		Other R/W Instructions	
Read the On-chip Port	Latch	Read the On-chip Port	Pins
CLR	bit	XCH	A,direct
SETB	bit	MOV	direct,direct
CPL	bit		
INC	direct		
DEC	direct		
ANL	direct,A		
ANL	direct,#data		
MOV	bit,C		
ORL	direct,A		
ORL	direct,#data		
JBC	bit,rel		
XRL	direct,A		
XRL	direct,#data		
DJNZ	direct,rel		

Tier 1 for the 80C324: Port 0 is not Bit Addressable in Tier 1. Port 2 is Bit Addressable.

Tier 1 Example

Figure 2 outlines the necessary blocks needed to implement Tier 1 Port Expansion Mode. This example builds three external ports.

Port-X allows execution of all possible PEM instructions (see Table 1) except "JBC bit, label," as previously noted. It requires both an output latch and an input buffer. Port-Y is provided as a Write-only port and requires just an output latch. Thus, instructions accessing this port address must be limited to those that are Write-only (see Table 1). Port-Z is a Read-only port and requires just an input buffer. Thus, instructions accessing this port address must be limited to those that are Read-only.

The external 2K-ohm resistor allows PXS to be sensed by the external logic.

PXS and ALE are used to decode the Address, Read and Write Strobes for the decode logic, buffers, and latches respectively.

The bit addressability of the ports is determined by the addresses assigned to the ports and whether or not read operations are externally implemented. Since all bit instructions require read operations, the Write-only Port-Y is excluded from bit addressability. Port-X and Port-Z can become Bit Addressable if assigned to addresses such as D8 and E8H. Bit-Addressable instructions used with Port-Z, however, would be limited to

those which are Read-only. Write operations to Port-Z are disregarded.

The choice of addresses C0, D8, E8, and F8H as port addresses (whether or not Bit Addressability is needed) is advantageous from a decoding standpoint. Since the next seven consecutive addresses beyond each of these locations are not defined in the 80C324, the address decoding may simply take place on the upper five bits of these addresses. The decoding is identical, whether or not Bit Addressability is desired.

The description of the Tier 1 example is independent of the choice of Hardwired or Software PEM.
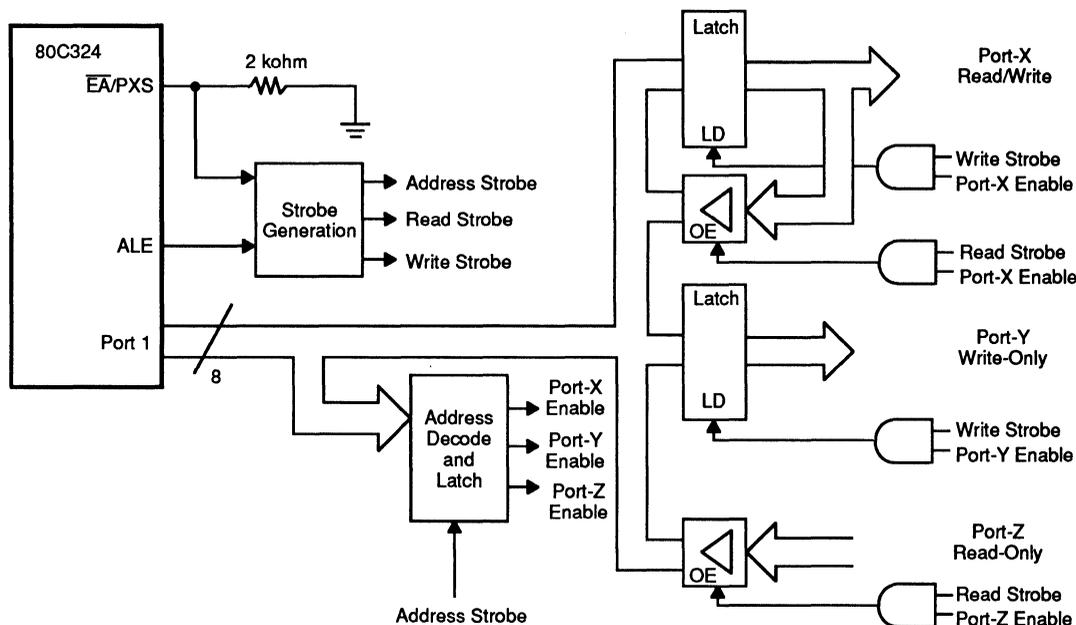
Figure 2. Tier 1 Logic Diagram

12837-002A

A Complete Approach—Tier 2

Tier 2 operation uses more than just PXS for decoding operations. It involves decoding the opcode of the instruction executing through synchronization with ALE and an external clock. The following capabilities are possible.

The user may build up to 14 External Ports with 7 being Bit Addressable. Bit addresses and byte addresses can be distinguished in the 91–97H range since the opcode is externally decoded. Port 1 can then be implemented as Bit Addressable without sacrificing port addresses 91–97H. This is the main difference between Tier 2 and Tier 1.

All instructions are now possible, if special attention is given to "JBC bit,label."

Read/Modify/Write implementation is possible if the RMW instructions are decoded as such and additional external logic is provided to read from either the external latch (output) or external buffer (input).

Tier 2 Example

Figure 3 outlines the blocks necessary to implement Tier 2 Port Expansion Mode. The Opcode-Decode block is shown, adding full Read/Modify/Write capability to the external ports, and allowing Port 0 to be Bit Addressable while still providing the additional ports 91 and 92H.

An external clock to the 80C324 is assumed, allowing a mechanism for latching the opcode. The first rising edge of ALE after reset indicates the beginning of instruction execution. The opcode is latched from Port 1 one clock cycle later (see Figure 1). The next opcode will appear either 12, 24, or 48 clocks later, as defined by the current opcode (four-cycle instructions "MUL" and "DIV" execute in 48 clocks). The PXS, ALE combination can still be used in Tier 1 to decode the Address, Read, and Write strobes.

If required, the "JBC bit,label" instruction may be implemented with the first read strobe coming from PXS, and the second read strobe coming from the Opcode Decode block 12 clocks later.

Tier 2 is very powerful. The user can actually implement his own custom operations based on correct decoding of the opcode and address data. For instance, the "MOV direct,A" instruction directed at Port Address F8H, for example, could be defined to broadcast the contents of the Accumulator to several external ports at once. Similarly, an "ORL direct,A" instruction, operating on a given port, could be defined to send its result to another external port. If the contents of the Accumulator were 00H before this operation was performed, a one-cycle "move" would result, increasing speed over the two-cycle "MOV direct,direct" instruction.

The description of the Tier 2 example is independent of the choice of Hardwired or Software PEM.
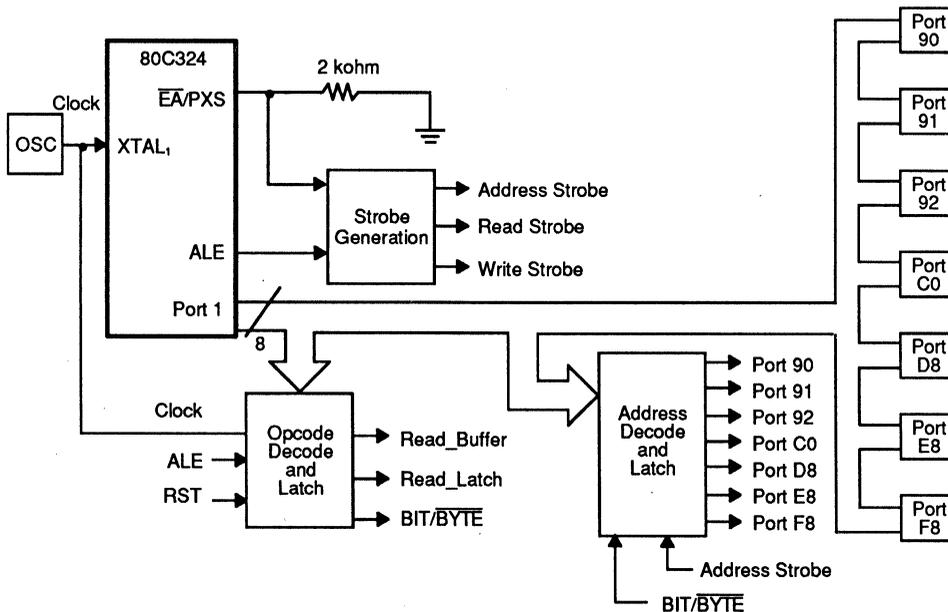
Figure 3. Tier 2 Diagram

ABSOLUTE MAXIMUM RATINGS

Storage temperature . . . . −65°C to +150°C

Voltage on any Pin to Vss . . . . . −0.5 V to Vcc +0.5 V

Voltage on Vcc to Vss . . . . −0.5 V to 6.5 V

Power dissipation . . . . 200 mW

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices

Temperature (TA) . . . . 0 to +70°C

Supply Voltage (Vcc) . . . . +4.5 V to +5.5 V

Ground (Vss) . . . . 0 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
V_{IL}	Input Low Voltage (except $\overline{EA}$)		−0.5	$0.2\,V_{cc}-0.1$	V
V_{IL1}	Input Low Voltage ($\overline{EA}$)		−0.5	$0.2\,V_{cc}-0.3$	V
V_{IH}	Input High Voltage (except XTAL$_1$, RST)		$0.2\,V_{cc}+0.9$	$V_{cc}+0.5$	V
V_{IH1}	Input High Voltage (XTAL$_1$, RST)		$0.7\,V_{cc}$	$V_{cc}+0.5$	V
V_{OL}	Output Low Voltage (Ports 1, 2, 3)	$I_{OL}=1.6$ mA (Note 1)		0.45	V
V_{OL1}	Output Low Voltage (Port 0, ALE $\overline{PSEN}$)	$I_{OL}=3.2$ mA (Note 1)		0.45	V
V_{OH}	Output High Voltage (Ports 1, 2, 3)	$I_{OH}=-60$ µA, $V_{cc}=5$ V ±10%	2.4		V
		$I_{OH}=-25$ µA	$0.75\,V_{cc}$		
		$I_{OH}=-10$ µA	$0.9\,V_{cc}$		
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode, ALE $\overline{PSEN}$)	$I_{OH}=-800$ µA, $V_{cc}=5$ V ±10%	2.4		V
		$I_{OH}=-300$ µA	$0.75\,V_{cc}$		V
		$I_{OH}=-80$ µA (Note 2)	$0.9\,V_{cc}$		V
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3)	$V_{IN}=0.45$ V		−50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1, 2, 3)	$V_{IN}=2$ V		−650	µA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	$0.45 < V_{IN} < V_{cc}$		±10	µA
RRST	Reset Pulldown Resistor		50	150	kohm
CIO	Pin Capacitance	Test Freq. = 1 MHz, $T_A=25$°C		10	pF
I_{PD}	Power-Down Current	$V_{cc}=2$ to 6 V (Note 3)		50	µA

<center>

Maximum Icc (mA)

</center>

Freq. VCC	Operating (Note 4)			Idle (Note 5)		
	4.5 V	5 V	5.5 V	4.5 V	5 V	5.5 V
3.5 MHz	6	8	10	1.5	2	3
8.0 MHz	11	14	18	2.5	3.5	5
12 MHz	15	20	25	3.5	5	6

Notes: 1. Capacitive loading on ports may cause spurious noise pulses to be superimposed on the V_{OL}S of ALE and other ports. The noise is due to external bus capacitance discharging into the port pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8 V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt-Trigger STROBE input. This note pertains to dual-in-line packages only. The additional V_{cc} and V_{ss} connections on the PLCC package from AMD removes this design consideration.

2. Capacitive loading on ports may cause the V_{OH} on ALE and $\overline{PSEN}$ to momentarily fall below the 0.9 V_{cc} specification when the address bits are stabilizing. This note pertains to dual-in-line packages only. The additional V_{cc} and V_{ss} connections on the PLCC package from AMD removes this design consideration.

3. Power-Down I_{cc} is measured with all output pins disconnected: $\overline{EA}$ = Port 0 = V_{cc}; XTAL$_2$ NC; RST = V_{ss}.

4. I_{cc} is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{ss} + 0.5 V, V_{IH} = V_{cc} − 0.5 V; XTAL$_2$ NC; $\overline{EA}$ = RST = Port 0 = V_{cc}. Typical values are approximately 50% lower. I_{cc} would be slightly higher if a crystal oscillator was used.

5. Idle I_{cc} is measured with all output pins disconnected; XTAL$_1$ driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{ss} + 0.5 V, V_{IH} = V_{cc} − 0.5 V; Port 0 = V_{cc}; $\overline{EA}$ = RST = V_{ss}, and the Watchdog Timer disabled.

Port Expansion Mode AC Timing
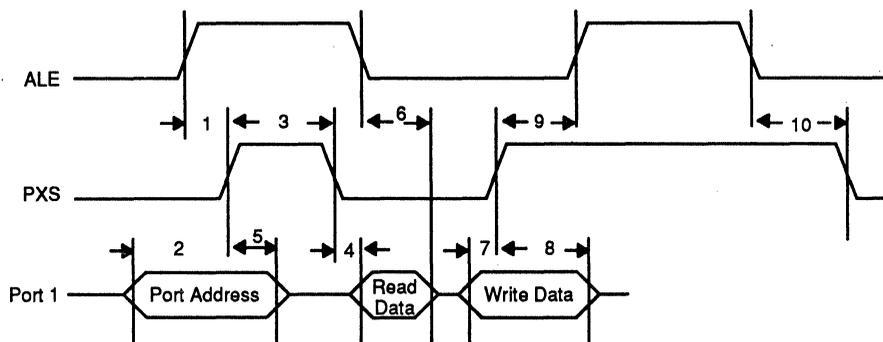
Port Expansion Timing

	Parameter Symbol	Parameter Description	Min.	Max.	Unit
	1/TCLCL	Oscillator Frequency	0.1	12	MHz
1	TAHPH	ALE High TO PXS High	10		ns
2	TAVPH	Address Valid to PXS High	-10		ns
3	TPPW	PXS Pulse Width	TCLCL-50		ns
4	TPLRD	PXS Low to Read Data Valid	TCLCL-50		ns
5	TPHANV	PXS High to Address Not Valid	TCLCL-25		ns
6	TALRDT	ALE Low to Read Data Tri-stated	0	TCLCL-10	ns
7	TWDPH	Write Data Valid to PXS High	TCLCL-50		ns
8	TPHWNV	PXS High to Write Data Not Valid	TCLCL-25		ns
9	TPHAH	PXS High to ALE High	TCLCL-30		ns
10	TALPL	ALE Low to PXS Low	TCLCL-35		ns

SWITCHING CHARACTERISTICS over operating range (C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF; C_L for all other Outputs = 80 pF)

Parameter Symbol	Parameter Description	12-MHz Osc. Min.	12-MHz Osc. Max.	Variable Oscillator Min.	Variable Oscillator Max.	Unit
EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS						
1/TCLCL	Oscillator Frequency			0.1	12	MHz
TLHLL	ALE Pulse Width	127		2TCLCL − 40		ns
TAVLL	Address Valid to ALE Low	28		TCLCL − 55		ns
TLLAX	Address Hold After ALE Low	48		TCLCL − 35		ns
TLLIV	ALE Low to Valid Instr. In		234		4TCLCL − 100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	43		TCLCL − 40		ns
TPLPH	$\overline{PSEN}$ Pulse Width	205		3TCLCL − 45		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr. In		145		3TCLCL − 105	ns
TPXIX	Input Instr. Hold After $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr. Float After $\overline{PSEN}$		59		TCLCL − 25	ns
TAVIV	Address to Valid Instr. In		312		5TCLCL − 105	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		10		10	ns
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL − 100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL − 100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		252		5TCLCL − 165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		ns
TRHDZ	Data Float After $\overline{RD}$		97		2TCLCL − 70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL − 150	ns
TAVDV	Address to Valid Data In		585		9TCLCL − 165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	3TCLCL − 50	3TCLCL + 50	ns
TAVWL	Address Valid to Read or Write Low	203		4TCLCL − 130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	23		TCLCL − 60		ns
TQVWH	Valid Data to Write High	433		7TCLCL − 150		ns
TWHQX	Data Hold After $\overline{WR}$	33		TCLCL − 50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	TCLCL − 40	TCLCL + 40	ns

80C324

CHAPTER 10

Third-Party Support Products

Advanced Micro Devices does not support, maintain, or guarantee the performance of third-party products described in this chapter.
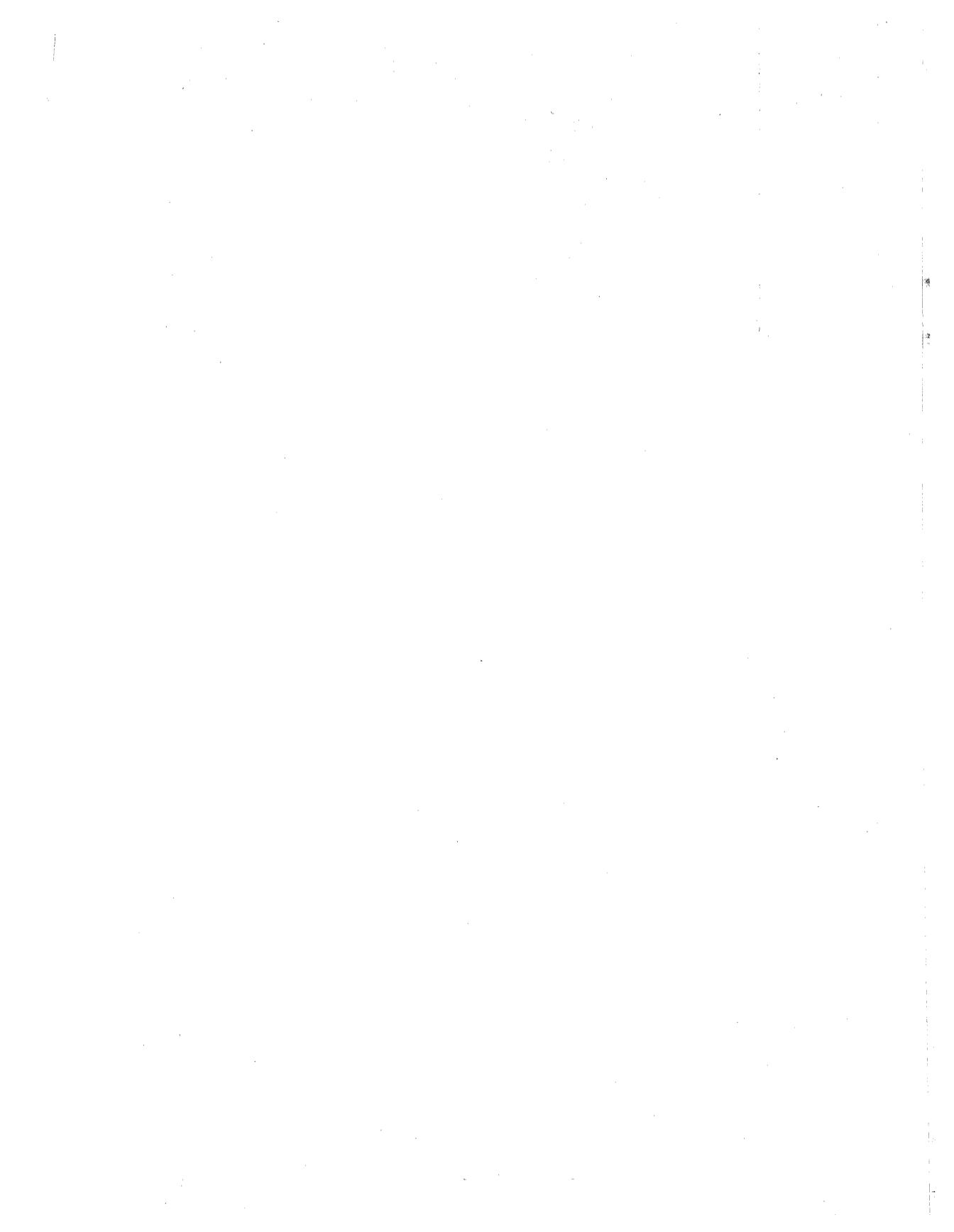
Chapter 10
Third-Party Support Products

INTRODUCTION

A number of support products are available for the 8051 microcontroller family. The following pages present product descriptions of emulators, assemblers, compilers, and programmers from various manufacturers. The material is intended to present a collection of what is available for AMD-manufactured 8051 Family microcontrollers, but is not necessarily a complete, up-to-date listing of all available products. Further information may be obtained from the individual companies listed and the many other vendors that support 8051 Family products. AMD does not guarantee the specifications of any of the products listed.

Third-Party Support Products

Vendor	Primary 8051 Family Products	Description
Hewlett-Packard 1501 Page Mill Road Palo Alto, CA 94304 (Contact local sales office)	Development System	Company provided, page 10-3
MetaLink Corporation PO Box 1329 Chandler, AZ 85244-1329 (602)926-0797 or (800) 638-2423	Development System	Company provided, page 10-8
American Automation 2651 Dow Avenue Tustin, CA 92680 (714)731-1661	Development System	Company provided, page 10-13
Huntsville Microsystems 4040 S. Memorial Parkway PO Box 12415 Huntsville, AL 35802 (205)881-6005	Development System	Company provided, page 10-14
Applied Microsystems Corp. 5020 148th Ave. N.E. PO Box 97002 Redmond, WA 98073-9702 (206)882-2000 or (800)426-3925 (U.S.) 44-(0)-296-625462 (U.K.)	Development System	Call vendor for details
Kontron Electronics D-8057 Eching/Munich Oskar-von-Miller-Str. 1 West Germany Phone: (0 81 65) 77-0	Development System	Call vendor for details
Nohau Corporation 51 E. Campbell Ave. Suite 107E Campbell, CA 95008 (408)866-1820	Development System	Call vendor for details
Signum Systems 1820 14th Street Suite 203 Santa Monica, CA 90404 (213)450-6096	Development System	Call vendor for details

Third-Party Support Products *(continued)*		
Vendor	**Primary 8051 Family Products**	**Description**
Sophia Systems NS Bldg 2-4-1 Nishishinjuku, Shinjuku-ku Tokyo 160, Japan 03-348-7000	Development System	Call vendor for details
Zax Corporation 2572 White Road Irving, CA 92714 (714)474-1170 or (800)421-0982	Development System	Call vendor for details
Franklin Software, Inc. 888 Saratoga Avenue #2 San Jose, CA 95159 (408)296-8051	C Compiler, Assembler	Call vendor for details
Micro Computer Control PO Box 275 Hopewell, NJ 08525 (609)466-1751	C Compiler, Assembler	Company provided, page 10-15
Archimedes Software 2159 Union Street San Francisco, CA 94123 (415)567-4010	C Compiler, Assembler	Company provided, page 10-20
Scientific Engineering Labs 255 Beacon St., Suite 3D Somerville, MA 02143 (617)625-0288	Pascal Compiler	Call vendor for details
Boston Systems Office 128 Technology Center Waltham, MA 02254-9164 (617)894-7800	PL/M Compiler, Assembler	Call vendor for details
Sysoft SA 6926 Montagnola Switzerland (091)543195	PL/M Compiler, Assembler	Call vendor for details
Cybernetic Micro Systems Box 3000 San Gregorio, CA 94074 (415)726-3000	Simulator, Debugger	Call vendor for details
Microtek Research Box 60337 Sunnyvale, CA 94088 (408)733-2919	Simulator, Assembler	Call vendor for details
Data I/O Contact local sales office or call: (800)247-5700 Dept 401	EPROM-version Programmer	Company provided, page 10-24
Stag Microsystems 1600 Wyatt Drive Santa Clara, CA 95054 (408)988-1118 or (800)227-8836	EPROM-version Programmer	Call vendor for details

HEWLETT-PACKARD DEVELOPMENT SYSTEM

Emulators

Hewlett-Packard offers a wide selection of emulators to support microprocessor and microcontroller-based product development. These emulators provide the essential link between software development and hardware/software integration. Code developed on the HP 64000 system or compatible host computers is executed on the emulation subsystem and user's target system, if available, for real-time debugging and logic analysis.

Hewlett-Packard emulators are part of an integrated set of design and development tools that include Teamwork/SA/RT/SD for structured analysis and design; cross compilers and assemblers/linkers for programming at the most efficient level; directed-syntax softkeys and an easy-to-use, responsive editor to streamline software development and documentation; and analysis subsystems which provide powerful measurements to investigate program execution, timing relationships, system performance, and processor activity.

Universal Development System

HP 64000 products comprise a universal development system that provides development support that includes the 8051 Family of microcontrollers. When additional emulators are introduced to support popular new processors, they are easily integrated with existing HP 64000 real-time analysis tools. This flexibility protects the capital investment in instrumentation, since new projects and goals can be accommodated with low-cost add-ons rather than total replacement of development systems and tools.

System Environment

Hewlett-Packard supports the universal development system with two system platforms; a general-purpose, multiuser computer and a dedicated, stand-alone workstation.

The HP 64000-UX Microprocessor Development Environment is based on the HP 9000 Series 300 general-purpose computer, running the HP-UX* operating system. This workstation platform is common to the design engineering tools of HP Design Center. The multiuser capability of the Series 300 allows for shared hardware and software resources among system users. Multiple window capability allows integration and debug tasks to be viewed simultaneously, for convenient observation of interactive debug information. The HP-UX operating environment supports user-programmable command files for repetitive and complex test routines. HP 64000-UX systems can be easily connected to other host computers or system resources.

The HP 64000-UX environment is compatible with the dedicated, stand-alone HP 64100A and 64110A Logic Development Stations. The same emulation and analysis card sets for most subsystems are used in both the HP 64000-UX Microprocessor Development Environment. In addition, these hardware platforms can be networked via high-speed link or RS-232 for maximum productivity.

Features

- Real-time emulation for evaluating target system performance and critical timing relationships
- Multiple emulation capabilities for multiprocessor product designs
- Display and modify memory, registers, and I/O ports
- Disassembly of microprocessor instruction set
- Source-line referencing
- Symbolic debugging for emulation and analysis operations
- Compatible and interactive high-performance logic analyzers for hardware, software, and software performance analysis
- Run control, single stepping, run from, and run until
- HP 64000 system resources (disc files, printer, development station keyboard, display, and RS-232 port) can be used to simulate target system I/O
- Emulation memory available from 32 Kbytes to 64 Kbytes
- Memory assigned by blocks to target system or emulation memory over the microprocessor's entire address space; designated as ROM, RAM, or illegal address space.
- User-definable emulator kit for custom emulation support

Measurement System Configuration

An HP 64000 emulation subsystem consists of an emulation control card, emulation pod, and operating software. An emulation bus analyzer is used for tracing activity on the emulation bus in real-time. Trace lists generated by the analyzer may be displayed in the mnemonics of the target processor. Inverse assembler software is included in the emulation software. HP 64856A User Definable Inverse Assembly software package may be used to generate mnemonics for the User Definable Emulator (UDE) and User Definable Preprocessor. Cross assemblers/linkers are available.

HP-UX is Hewlett-Packard's implementation of the UNIX operating system.

The analytical functions of the emulator can be expanded with Model 64310A Software Performance Analyzer. Input data from the HP 64310A analyzer is collected from activity on the emulation bus. The performance analyzer provides the macro overview measurements needed for optimizing and modifying code for more efficient software performance.

When complex, detailed logic state analysis is required, the powerful HP 64620S Logic State/Software Analyzer can be integrated directly into the emulator subsystem via HP 64304A Emulation Bus Preprocessor. The added power of software analysis provides traces converted to high-level language source code as well as assembly language or numeric code lists.

For hardware debugging, the powerful HP 64610S High-speed Timing/State Analyzer checks timing relationships, locates glitches, and identifies marginal signals. For high-speed logic designs, the analyzer functions as a 125-MHz state analyzer.

A new dimension of analysis power can be added with the Intermodule Bus (IMB) which links analyzers and emulators. The IMB communicates with the emulator through the HP 64302A Emulation Bus Analyzer. Other analysis subsystems that can be added to the IMB are the HP 64620S Logic State/Software Analyzer, HP 64310A Software Performance Analyzer, HP 64610S High-speed Timing/State Analyzer, and HP 64340 Real-time High-level Software Analyzer. Cross triggering between analyzers enables the designer to make coordinated measurements that help solve complex hardware/software integration problems.

System Architecture

All emulators of the HP 64000 system use a multiple-bus architecture, thus allowing interactive emulation and analysis. The development station host processor communicates with all installed subsystems using the HP 64000 system bus. A separate high-speed emulation bus
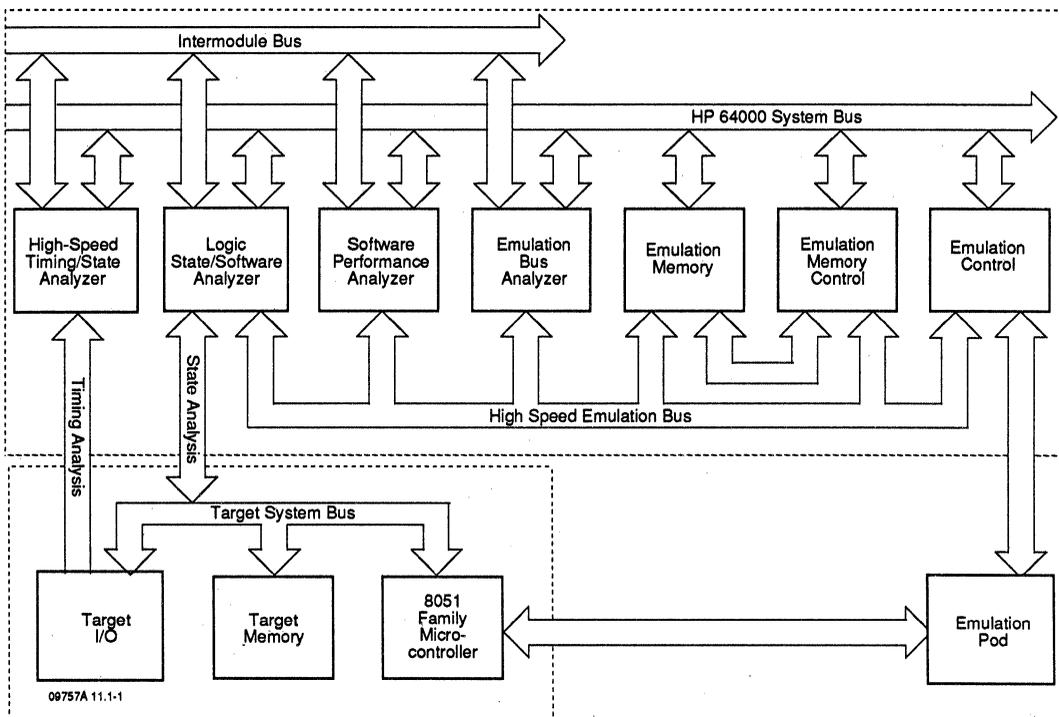
Figure 10-1. System Architecture

carries all transactions required for emulation. Independent operation frees the emulation system from the host system overhead. The intermodule bus controls sophisticated, interactive cross measurements for emulation, state, timing, and performance analysis. Major advantages of the multiple-bus architecture are real-time, transparent emulation and analysis that free the target system for unrestricted execution.

8051/8751/8031/8053/8753 Model 64264S

Model 64264S Emulation Subsystem consists of a control board, pod, and software. Connection to the target system is made with a 305 mm (12 in.) cable that terminates in a 40-pin, low-profile probe. A typical 8051/8751/8053/8753 emulation system includes HP 64264S Emulation Subsystem, HP 64156S Emulation Memory System, and HP 64302A Emulation Bus Analyzer.

Software development support is provided by Model 64855 Cross Assembler/Linker.

Features

- Real-time execution up to 12 MHz independent of emulator/target system memory assignment
- Nonintrusive, real-time traces of 8051 activity for basic analysis and evaluation including access to
  - Program memory
  - Internal and external data memory
  - Accumulator and special-function registers
  - I/O ports 0, 1, 2, and 3
- Disassembly of 8051 instruction set
- Program and external data memory mapped in 256-byte blocks to emulation or target system memory
- Expanded measurements capabilities through interactive operations with other HP 64000 subsystems:
  Another 8051 emulator or any other HP 64000 emulator
  HP 64620S Logic State/Software Analyzer
  HP 64610S High-speed Timing/State Analyzer
  HP 64310A Software Performance Analyzer

Electrical Specifications

Maximum clock speed: 12 MHz

Inputs: all inputs meet AMD specifications plus approximately 40 pF capacitance; Port 0, low-level input, 0.45 mA; Port 1, Port 2, and Reset, low-level input, 0.1 mA; and EA, low-level input, 0.5 mA.

Power: 20 mA drawn from the target system; all other power supplies by the development station or card cage.

8051 Cross Assembler/Linker

The HP 64855 Cross Assembler/Linker provides assembly language software development support for the 8051 Family of Microcontrollers. The Model 64855AF is hosted on the HP 64100A/64110A development stations. Model 64855S and the appropriate option provide a cross assembler/linker which executes on both the HP 64100A/64110A development stations and on an HP 9000 series 300 HP-UX or a VAX/VMS host computer system.

Regardless of the host computer execution environment, the cross assembler/linkers produce identical relocatable and absolute code for a given source program. The assembler uses the instruction mnemonics for the 8051 series and generates code for all the defined 8051 instructions. However, due to differences in some pseudo instruction mnemonics and assembler syntax conventions, source programs written for the manufacturer's assembler generally require some modification prior to use with the HP 64855 Cross Assembler/Linker.

Both assemblers/linkers generate the necessary information for symbolic debug in emulation. Programmers can troubleshoot the code using source program line numbers and global symbols, eliminating the task of looking up addresses.

Assembler Directive

"8051" causes the cross assembler/linker to recognize the instruction set of the 8051 Family of Microcontrollers.

Reference Information

Addressing/Operand Field Conventions

8051 Registers — The 8051 microprocessor contains 128 bytes of on-chip RAM (expandable to 65,536 bytes with external RAM chips). Addresses 00H to 1FH in RAM are reserved for 32 general purpose registers arranged in four register blanks; R0-R7 indicate the eight working registers; these registers are called the current active bank. The current active bank can be changed to any of the other register banks by specifying the register bank select bits RS0 and RS1 in the program status word.

The stack is also located in the on-chip RAM and the Stack Register points to the top of the stack. On RESET, the stack pointer is set to 07H. The Stack Register cannot exceed 127 (7FH in hex).

There are additional hardware registers for the 8051 which are located on an external RAM chip. The registers and their addresses in external RAM are shown on the following page.

External RAM Registers and Addresses

ACC	Accumulator	0E0H
B	Multiplication	0F0H
DPH/DPL	Data Pointer High/Low	83H/82H
IE	Interrupt Enable	0A8H
IP	Interrupt Priority	0BDH
P0-3	Ports 0-3	80H, 90H, 0A0H, 0B0H
PSW	Program Status Word	0D0H
SBUF/SCON	Serial Buffer/Control	99H/98H
SP	Stack Pointer	81H
TCON/TMOD	Timer Control/Mode	88H/89H
TH0/TL0	Timer 0 High/Low-Byte	8CH/8AH
TH1/TL1	Timer 1 High/Low Byte	8DA/8BH

The HP 64855 Cross Assembler/Linker supports all five addressing modes of the 8051 microprocessor: Immediate, Data, Indirect, Bit, and Code Addressing. The addressing modes are as follows:

Immediate Addressing — Any number, symbol, or expression may be specified as an operand by immediately preceding it with a pound (#) symbol. Examples:

   #number, #symbol, #expression, #"ASCII char"

Data Addressing — Data can be obtained from any of the 128 on-chip RAM addresses or a hardware register address. (External RAM data must be obtained by indirect addressing.) The symbol or numeric expression must be of either no segment type or type DSEG (ie., previously defined to be within the data segment). Data addresses from 0-127 are in RAM and addresses from 128-255 are in hardware registers.

MOV   A,76H   ;Move contents of address 76H to accumulator.

Indirect Addressing — The address of the operand is pointed to by register R0 or R1 in the active register bank if the indirect address is in on-chip RAM. External code or data memory is addressed by the MOVC or MOVX instructions by using the Data Pointer Register (DPTR). The address within R0 or R1 must be between 0-127. The indirect mode is specified by preceding the register with a (@). For example:

ADD   A,@R0   ; Add contents of the on-chip RAM
                   ; Address in R0 to accumulator.
MOVC R0,@DPTR; If DPTR contains 1000H, then
                   ; move the data at address 1000H
                   ; to register R0.

Bit Addressing — The processor can access any bit in the on-chip RAM and other hardware registers. The byte which contains the bit must be defined, followed by the bit

selector (.) and the bit identifier (0-7). Opcodes using a bit address must be defined as type BSEG or no segment type. For example:

SETB 5CH.3   ;Set bit 3 at address 5CH.

Code Addressing — The instruction specifies a new location to jump to in the program code.

Pseudo Instructions

The HP 64855 Assembler/Linker recognizes most of the basic ASM51 Assembler pseudo instructions as have equivalents for many of the others. The following lists the pseudo instructions that are similar to the HP assembler pseudos.

ASM 51 Assembler Pseudos	HP 6400 Equivalent Assembler Pseudos
EJECT	SKIP
END	END
EQU	EQU
IF..ELSE..ENDIF	IF..ELSE..ENDIF
MACRO..ENDM	MACRO..MEND
ORG	ORG

The HP 64855 Cross Assembler/Linker also supports several additional pseudo instructions for the 8051 processor which differ from the general HP 64000 assembler pseudos.

BIT   BIT assigns a bit address to a symbol. This allows the assembler to refer to a specific bit.

BSEG   This selects all data to be in the bit address segment. The locations in the bit address segment must be within the range from 0–255.

CSEG	CSEG invokes the program relocatable counter. (This is default when the assembler is invoked.) The counter can range from 0-65,535.
DATA	DATA assigns an on-chip address to the symbol. The symbol is defined as type DSEG.
DB	Stores data by types in consecutive memory locations within the code segments starting at the current setting of the program counter.
DBIT	DBIT reserves bit address space.
DS	This reserves or defines a block of space by types in any segment in memory.
DSEG	DSEG selects the on-chip data address segment. Addresses range from 0-255.
DW	Stores data by words in memory. DW is only valid within the CSEG or code segment.

SET	The SET pseudo is the same as the HP 64000 pseudo; however, the HP 64855 assembler also uses SET to assign a name to one of the 8051's registers.
XDATA	XDATA assigns an off-chip data address to a symbol and makes the symbol type XSEG.
XSEG	XSEG selects the external data address segment. The values in the location counter range from 0-65,535.

The HP 64855 Cross Assembler/Linker does not support the following HP 64000 pseudo instruction; the alternate pseudo must be used instead.

DATA	Use XSEG or DSEG instead.
DEC	Use DECIMAL instead.

Special Note:

Hewlett-Packard has just announced a brand new series of low-cost, host-independent emulators. The new HP 64700 Series emulators can be connected to a variety of hosts including the HP 9000 Series 300 and the PC. Please contact your Hewlett-Packard Sales Representative for an up-to-date list of supported processors.

METALINK DEVELOPMENT SYSTEM

The MetaLink Emulator, What is it?

An In-Circuit Emulator is a tool for use in designing systems incorporating microcontrollers. Using this tool, the system designer can interactively control and examine the state of the system at any chosen time. This is essential for speeding up the debugging process and enhancing the system designer's productivity. The tool is easy to use; simply replace the system microcontroller with the emulator probe, which then becomes the in-circuit microcontroller. When the probe is connected to the host computer, the system can be completely controlled.

The emulator provides not only the capabilities of the target processor, but a set of debugging capabilities to facilitate and shorten the debugging process. Why is this important? It is not enough for the emulator to simply behave like the target processor, it must also provide read/write access to all signals and all data to which the microcontroller has access. This includes information which resides inside the microcontroller. Without this access, the engineer may *not* be able to completely control and debug the system.

The many uses of the emulator can be easily visualized after examining a typical system design cycle.

The first use of an emulator in the design cycle is in the software-development phase. The emulator executes the program exactly as the target system would, in real time, and it provides all of the interactive debugging capabilities. Software, developed using the emulator, can be completely debugged, except for the hardware interface, before it is integrated with the system hardware.

The second and major use of an emulator in the design cycle is in the integration of the target software and the system hardware. Even when the hardware and software have each been individually debugged, new problems can surface when they are joined together. The emulator is used, in this case, to solve these potential problems.

After a prototype has been completely debugged, the emulator can then be used to test the specs of the system. Worst case parametric tests can be developed and tested on the prototype. This provides the designer with valuable information about the limitations of the system. It also provides test programs which can be used in the manufacturing process (see below).

The third use of an emulator is in the product-manufacturing phase. The same test routines, used to develop and debug the prototype, or even more comprehensive test routines, can be used to test the finished products. Any non-functioning units can be easily debugged using the emulator's full range of debugging capabilities.

PHASE   **TOOLS USED**

- Conception
- Architecture → RTL Simulator
- Logic → Logic Simulator
- Software → Emulator/Simulator
- Hardware Prototype → CAD/CAM Tools
- Integration → Emulator
- Manufacturing Testing → Emulator/Tester
- Field Testing → Emulator/Tester
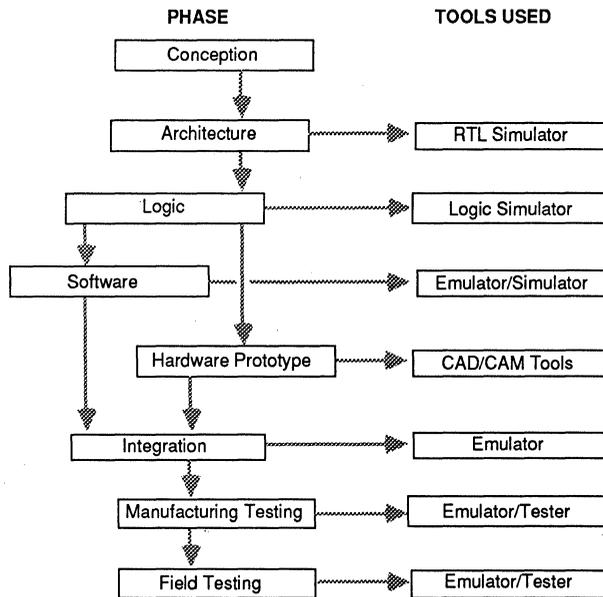
Figure 10-2. Design Cycle

The fourth use of an emulator is in the field-service phase. The MetaLink emulator can run on any IBM PC or 100% PC-compatible host computer including the PC-compatible portables. Check the end of this description for other operating systems and host computers compatible with the MetaLink emulator. If the field location already has a host computer, the field service team need only carry the emulator module,which easily fits in a briefcase, and some floppy disks. If a host computer is not available, a portable host can be used.

MetaLink MetalCE or MicroICE Emulator

The MetalCE or MicroICE emulator is a PC-based in-circuit emulator, designed for use in developing, testing and debugging designs based on the 8051 Family of single-chip microcontrollers. Using the MetalCE or MicroICE emulator, hardware and software designs can be developed simultaneously. The MetalCE or MicroICE emulator assists in the following phases: software development, integration of target software and system hardware, manufacturing and field service.

The MetalCE or MicroICE emulator may be used with several third-party software cross-assemblers and compilers in the development phase that in the integration phase can also provide symbolic debug capability. These are:

- Cross-Assemblers - MetaLink's, IAR Systems, Enertec, Microtec Research and Intel.

- Compilers - Archimedes Software, IAR Systems and Intel.

Significant features of the MetalCE or MicroICE emulators:

- Serially linked to IBM PC or compatible hosts

- Advanced menu-driven human interface

- Real-time and transparent emulation up to 16 MHz

- Disassembler and single-line assembler

- Examine/modify memory capabilities

- 16 break and trace-trigger conditions

- High Level Language Support

- Supports both modes:
  - Microprocessor
  - Microcomputer

- 9 probe clips
  - 7 External events
  - 1 External trigger input
  - 1 External trigger output

- Up to 128,000 break and trace triggers

- Emulation Memory:
  - 64K Program
  - 64K External data

- Full symbolic debug capability

- Opcode class editor

- Up to 64K pass counts

- Separate program and data-memory mapping in 16-byte blocks

- Experiment editor/compiler

- Trace with 4K frames (MetalCE)
  Trace with 2K frames (MicroICE)
  - Start, end and center triggers

Emulator Functions

Various MetalCE or MicroICE emulators can support different versions of the 8051 Family of microcontrollers. See Table 10-1. They will support NMOS and CMOS versions of the devices, up-to a clock rate of 16 MHz, where appropriate. The MetalCE or MicroICE emulator is totally transparent to the users target system and will function at the clock rate specified by the user.

The MetalCE or MicroICE emulator functions from an IBM PC or compatible computer and is controlled by the serial-interface board of the system. The serial-interface operation rate is controlled by the user and the target-system clock rate; 9600 bps is the maximum transfer rate.The user interacts with the keyboard and the PC screen, while the PC's RAM memory provides the resident home for the MetalCE or MicroICE application system and user target program.

Table 10-1. MetalCE or MicroICE Emulator Part/Model Number Listing

Part Number	Model Number	AMD Devices Supported
MC-8031	MicroICE-8031	8031 & 80C31
MC-8052	MicroICE-8052	8031, 8751, 8753, 8051, 8053, 80C31 & 80C51
MI-80515	MetalCE-80515	80515, 80535
MI-80C521	MetalCE-80C521	80C521, 80C321 & 80C541
MC-80C321	MicroICE-80C321	80C321 & 80C31
MI-80535	MetalCE-80535	80535

User Interface

The MetalCE or MicroICE system uses a menu driven screen format for commands; a menu is structured as follows:

Command1 Command2 Command3
Quick help description of Command1

MENU NAME

Errors, warnings or messages

The first line of the screen contains a list of the command options available for that menu. The second line contains a one-line description of the highlighted command (see below). The middle of the screen contains the menu's name. The line at the bottom of the screen contains any errors, warnings or messages encountered during a command execution.

User Abilities

The MetalCE or MicroICE emulator can perform the following functions and call the following sub-functions:

- Load program code memory from disk files
- Upload program code memory from user target system board
- Download user board external data memory from disk files
- Call the system-configuration menu
- Restore a previously saved system and status
- Store the system and status in a disk file
- Create or execute a macro command file
- Call the interrogate menu
- Call the Help menu
- Terminate a session
- Escape out to and return from the resident operating system

User Interface Selection

The user selection specifies the baud rate used and the communications port (1 or 2) used for communication between the MetalCE or MicroICE emulator module and the host computer. It also includes the mode of operation and the configuration of the external data bus. Most MetalCE or MicroICE emulator models give the user the option to select between External Address Bus Mode (ROMless) and Single-Device Mode (ROM) with various external program/data memory addresses and all or some of the I/O ports.

Interrogation Selection

The Interrogate portion of the MetalCE or MicroICE emulator allows the user to run emulation experiments against the target system, to examine the status of the system, to set break and trace triggers and to examine/modify data, using the following capabilities:

- Running an emulation experiment
- Single stepping the target
- Resetting the target
- Setting a phantom breakpoint then running an emulation
- Setting up to 16 simple breakpoint/trace triggers or ranges
- Setting the repetition counter
- Setting the trace-trigger type (Start, Center or End)
- Calling the Help menu
- Examination and modification of SFRs and registers
- Examination and modification of internal data memory
- Examination and modification of external data memory
- Examination and modification of code memory
- Viewing the 2K or 4K trace buffer
- Examination and modification of the emulation experiment
- Selecting the 7 probe clips for trace
- Setting up to 16 increment pass-count addresses or ranges
- Escape out to and return from the resident operating system
- Viewing the A/D conversion data
- Turn Trace Trigger ON/OFF (MicroICE)

Experiment Selection

An experiment is the specification of where breakpoints, trace triggers or counts are to occur. It can be described in high-level language, called the Experiment Language, using the MetalCE or MicroICE emulator software. An experiment,then, is simply the Experiment Language text that describes where the breakpoints are to occur. Up to 128,000 complex hardware breakpoints, trace triggers or counts can be set in the MetalCE or MicroICE emulator.

An experiment can be created outside the MetalCE or MicroICE environment by using any available text editor to create an experiment text file. This file can then be read into the MetalCE or MicroICE system and then interacts with the user program to cause those breakpoints, trace-triggers and counts to occur.The experiment uses the *If-then* condition statement as its basic construct. Experiment statements will be of the form:

if (condition) then (action).

The condition represents a breakpoint or trace-trigger specification. Breakpoints or trace-triggers can be specified by any of the following methods:

A PC address

A PC address range

An opcode value

An opcode class

A direct byte address

A direct byte address range

A direct bit address

A direct bit address range

An immediate operand value

A read or write to bit or direct address

An external data address

An external data address range

Logical AND or OR of the above

Pass count overflow

External Input

The action represents the type of event that will occur after the condition has been encountered. The type of action can be specified by any of the following:

A break

An enable/trace

A count

A count/output trigger

In addition, an Examine/Modify Experiment Editor exists that can be used to examine and modify an experiment specification. In this editor, the user can:

Edit an experiment

Compile an experiment to set the breakpoints

Load an experiment from a disk file

Store an experiment in a disk file

Reset the current experiment

Delete the current experiment

Call the Opcode Class experiment

Examine/Modify Memory

Using the MetalCE or MicroICE emulator, the user can examine and modify the five memory spaces of the 8051 Family of devices. This examination/modification of memory spaces is broken down into two areas: Program-Code memory and Data memory.

Using the Examine/Modify Program Code Memory is used to examine and modify the contents of the MetalCE or MicroICE emulator code memory and provide for the following functions:

Disassembly of the program code (hex or symbolic data)

Single-line assembly of the program code

Examination and modification of raw program-code memory data

Examination and modification of program-code memory mapping

Selective mapping of the 64K program-code memory to the emulator

Selective mapping of the 64K program-code memory to the user

The Examine/Modify Memory Data is used to examine and modify the contents of the MetalCE or MicroICE emulator internal-data memory, the external-data memory and the MetalCE or MicroICE emulator table memory. It allows:

Dumping a block of memory content

Scanning and modifying each memory, a byte at a time

Filling a memory block with data

Moving a block of memory content from one location to another

Searching each memory for a data pattern

Verifying and comparing one block of memory data with another

Examining and modifying the directly addressable bits, which are mapped to the internal-data memory space

Selective mapping of the 64K external-data memory to the emulator system

Selective mapping of the 64K external-data memory to the user system

Macro Capabilities

The Macro is used to create and execute macro command files. A macro command file contains groupings of MetalCE or MicroICE commands which, when executed together, perform a macro function. These macro functions are typically repetitious tasks that are performed over and over again in one or many debugging sessions. Using the macro-command facility, the designer can define the macro-command file *once* and then execute it anytime later in the same or even another debugging session.

Symbolic Debug

The MetalCE or MicroICE emulator supports user and pre-defined symbols. The use of a name and not an address can alter the content of bits, bytes and code. In addition, five different object-file formats are accepted: standard Intel hex-file format; Intel absolute-object-module format; Microtek Research absolute-output-ob-

ject modules; IAR, Enertec, Archimedes object modules and MetaLink absolute-object-file format. Standard Intel hex-files can be created by assembling the user's program code with most of the currently available MCS-51 cross assemblers. Intel object-module files can be created by linking/locating modules with Intel's RL51 program. These source modules can be either assembled ASM51 object modules or compiled PLM object modules. MetaLink absolute object files are created by the MetaLink ASM51 Macro Cross Assembler.

System Requirements

Hardware Requirements

- IBM PC or a compatible PC
- Two 5-1/4 in. double-sided/double-density floppy disk drive
- 640K bytes of memory
- RS232C interface board

- RS232 cable with a male connector at the emulator end.
- Emulator power supply
  1.5 A + 5 VDC ± 5% (MicroICE)
  1.0 A + 23 VDC ± 5% (MetalCE)

Software Requirements

- PC DOS version 2.0 or later

High Level Language Support

The MetalCE or MicroICE emulator supports PLM and 'C' language compilers with advanced line number and multi-module capabilities. Line numbers, procedures and multi-module labels may be used for a number of emulator operations including: trace triggers, disassembly, fit, pass counts, etc. Using the MetalCE or MicroICE emulator, the user has the ability to single-step by machine instruction, procedure, line number in the current module, or line-number access all modules.

AMERICAN AUTOMATION DEVELOPMENT SYSTEM

EZ-PRO 2.1 Development System

American Automation's EZ-PRO 2.1 Development System is a complete development environment for microprocessor-based systems. Supporting the 8031/8051 family of microcontrollers (and over 70 other microprocessor models), EZ-PRO's integrated tools help to implement and debug microcontroller designs. The system includes the following:

Cross-assemblers with programmable macro expressions

Relocating linkers with user-library support

K&R standardized C-language cross-compilers

Exceptional symbolic debuggers

Fast in-circuit emulators

EPROM programming utilities

Flexible EPROM programming hardware

File conversion utilities

American Automation's in-circuit emulators and associated symbolic debuggers form the heart of the EZ-PRO system. The emulators feature transparent, non-invasive emulation with no wait states. Integrated break-pointing and bus-tracing tools pinpoint problems while the interactive assembly/disassembly facility helps to examine and modify the code under test. Using a flexible memory-mapping scheme, software may be tested in any combination of target system and emulator memory; software may also be tested without any target system attached.

The powerful development hardware is backed by an equally powerful suite of software development tools: C-language cross-compilers, macro cross-assemblers, relocating linkers, and the symbolic debugging package. Each package contains several exceptional features.

The C cross-compilers feature rapid compilation and generate tight, fast code. Extensions to the basic C-compiler support the 8051's special features, and 8051-series users may select from one of four memory-saving models, designed to fit generated code into even the tightest of spaces. Assembly-language modules may be intermixed with C modules for even greater speed and compactness.

The EZ-PRO Macro Relocatable Cross-Assemblers feature not only a powerful "macro expression language", but also support a wide range of pseudo-operations. Each assembler conforms exactly to the manufacturer's standard mnemonics.

aaLINK, the EZ-PRO Relocating Linker, assembles output modules from several sources — including the EZ-PRO assemblers and C cross-compilers — into a final executable module. The final output file may be easily modified by changing a command file.

Finally, the tested software may be placed into an 8751-series microcontroller or 27XX-series EPROM using the integrated EPROM programming tools. This, the EZ-PRO system provides a complete development environment.

The emulators connect to a host computer through an RS-232C link. Their modular design permits upgrades both to support new microprocessors and to add new features and extended memory. The 8051-emulator features are listed below.

- A complete symbolic debugging facility
- Advanced breakpointing features
- Fast menu-driven system
- Operates at full clock speed with no wait states
- Fully transparent emulation — all resources available to target system
- 4K Deep Trace (tm) includes trace management
- Complex triggering features include ranging, pass counts, and sequential breakpoints
- Performance analysis tools
- Memory-conserving C cross-compiler
- Macro relocatable cross-assembler
- EPROM programmer supports 8751 series
- Supports NMOS, CMOS, and EPROM versions
- TeleService extended service and TelePresence remote diagnostics available
- Host systems include the IBM PC and IBM-PS/2 series, Sun 3 Workstations and Macintosh development software and systems
- 5-year warranty

American Automation backs each EZ-PRO system with superior customer support. This support includes a 5-year warranty, telephone support, software updates, and the TeleService extended service plan.

HUNTSVILLE MICROSYSTEMS DEVELOPMENT SYSTEM

Huntsville Microsystems has two low-cost in-circuit emulators that support the 8051 family of microcontrollers, the SBE-51 and the SBE-31. Both emulators support 16 MHz real-time emulation from either an internal (emulator) clock or an external (target system) clock or crystal. They also contain a real-time trace, five hardware breakpoints, an in-line assembler and a disassembler; either emulator can be run from a host computer or a dumb terminal. A Relocatable Macro-Cross Assembler and a Symbolic Debugger are available for the IBM PC family computers and compatibles. See Table 10-2 for available emulator packages.

The SBE-51 supports the internal or on-chip program memory versions of the microcontroller such as the 8051, 8751H, 8753H and 8053. It is a true on-chip program-memory emulation and does not require the use of any of the four I/O ports. Thus, the user has exclusive control of all four of these ports. The SBE-51 is a non-intrusive in-circuit emulator and does not use or restrict any of the microcontroller's functions. The unit contains 16K bytes of on-chip program memory (much larger than the 4K bytes of the 8051 or the 8K bytes of the 8053) providing the user with the capability to download much larger programs during the development cycle. The SBE-51 also supports the CMOS version including functions such as idle and power-down mode.

The SBE-31 supports the external or off-chip program memory version of the 8051 family microcontrollers, such as the 8031AH and 80C31BH. The unit contains 64K bytes of emulation memory that may be used for external program or external data memory. The user's target-system memory may be added to the emulator's memory to complete the 128K byte address space (64K byte program memory, 64K byte data memory). The SBE-31 will also support CMOS designs and CMOS functions.

Features

- Real-time emulation up to 16 MHz with five hardware breakpoints and single step.
- 500 cycles of real-time trace history.
- 16K bytes of program memory (SBE-51)
- 64K bytes of memory, mappable in 2K blocks between program and data memory (SBE-31)
- RS232C interface can operate with a terminal or can be slaved to a host computer.
- Examine/modify memory, registers, flags, timer/counters, I/O ports, stacks and program counter.
- In-line assembler and disassembler.
- Uses internal oscillator or external oscillator or crystal.
- Upload or download Intel hexadecimal files.
- Complete software and hardware debugging facilities.
- Powerful command set includes fill-memory block, move-memory block, compare-memory blocks and test-memory blocks.
- Relocatable Macro-Cross Assembler and Symbolic Debugger for the IBM-PC, XT, AT and compatibles and all CP/M systems.
- Symbolic debugger for PL/M51 and assembly language.

Table 10-2. Emulator Packages

Microcontrollers Supported	Description of Emulator Package	Part
8051AH, 8751H, 8053, 80C51BH, 8753H	Complete development package for IBM PC family computers (includes all five items described below):	IDP-51
	1. 16 MHz Single Board Emulator for 8051 family on-chip internal program memory microcontrollers	SBE-51
	2. Relocatable macro Cross Assembler for IBM PC family computers	HMA-51R
	3. Symbolic Debugger Communications package for IBM PC family computer	SBE-LS51
	4. Power supply for Single-Board Emulator	SBE-PS1
	5. Computer-to-Emulator interface cable (RS232 - Specify if other than male/female cable)	SBE-IC6
8031AH, 80C31BH	Complete development package for IBM PC family computers (includes item 6 below and items 2-4 above)	IDP-31
	6. 16 MHz Single Board Emulator for 8031 off-chip external program memory microcontrollers.	SBE-31
ALL ABOVE	HMI-200 Series Advanced In-Circuit Emulator for the 8051/8031 family. "C" and PL/M51 source level debugger available.	HMI-200-8051

MICRO COMPUTER CONTROL
8051 C COMPILER

General Description

MICRO/C-51 is an MS-DOS based C-like language cross-compiler for the 8051 family of single-chip micro-controllers, including the 80C521 and 80515. It is designed to provide access to all hardware resources of memory maps, interrupts, all on-chip peripherals, and the Boolean processor directly from C.

MICRO/C-51 supports a number of important features that provide direct access to the 8051 architecture:

- Assignment of variables to any of the five 8051 memory maps.
- C-pointer support for all 8051 memory maps.
- Direct C-source access to all special-function registers by name.
- C-source-level handling of 16 hardware-interrupt sources.
- Fast-interrupt context switching to any one of four register banks.

Chip Features Supported

Object Memory Maps

b-map – on-chip bit addressable (128 bits)

d-map – on-chip direct access (128 bytes)

i-map – on-chip indirect access (128/256 bytes)

p-map – external page zero (256 bytes)

e-map – external data (64K bytes)

c-map – external code (64K bytes)

Special Function Registers

Direct C access to all special-function registers by name.

Boolean Processor

Direct C access to on-chip bit map and all bit-addressable special-function registers by name.

Interrupts

Drive any C function directly from any interrupt source.

Fast interrupt context switching to any one-of-four register banks.

Run-time Features Supported

Math and memory-map exception handling.

Expandable pointer access to external memory-mapped hardware devices.

Compile Time Options

Default object-memory map selection

Listing control options

Debug support
Function trace
Stack monitor
Statement labels

Compiler Output

Assembly-language source file compatible with MICRO/ASM-51 relocatable macro assembler. Linkable with user generated assembly or PL/M-51 source files.

C Language Features Supported

MICRO/C-51 V1.0 is a subset implementation of the C language as documented in "The C Programming Language" by Kernighan and Ritchie. Processor specific extensions have been added to support micro-controller hardware resources.

Comments (/*...*/)

Identifier names (8 characters)

Constants
Integer (decimal, octal, hex)
Character ('x')
Escape (\a,\b,\f,\n,\r,\t,\v,\\,\',*,\",\ddd)

String ("string")

Declared-object types
bit – 1-bit unsigned (K&R Extension)
char – 8-bit signed
int – 16-bit signed
ptr – 24-bit unsigned pointer to char or integer
array – single dimensioned char or integer array
func ( ) – function return value
Options:
Interrupt driven
Using specified register bank

Storage Classes
extern – reference to externally declared object
global – objects defined outside a function
local – objects defined within a compound statement
static – restrict global object scope.

Statements

compound	case
if, if-else	default
while	break
do-while	continue
for	return
switch	null
expression	

Operators

Unary (*, &, -, !, ~, ++, - -)

Multiplicative (*, /, %)

Additive (+, -)

Shift (<<,>>)

Relational (<,>,<=,>=)

Equality (= =, ! =)

Bitwise (&, ^, |)

Logical (&&, | |)

Conditional (?:)

Assignment (=)

Comma (,)

Preprocessors

Conditionsl (#if-#endif up to 16 nested levels)

Include Files (#include up to 8 nested levels)

Macro Definition (#define text replacement)

Separate Compilation and Linkage

Library Functions

getchar, putchar, printf, etc.

Special run-time debug functions supporting debug operation via the on-chip serial port.

Compiler Operation

Input/Output

MICRO/C-51 accepts, as input, a C source file created with a standard text editor. This file must have the extension (.c). Assembly-language source output is sent to "filename.src". Listing and error messages are sent to the MSDOS standard output file, normally the console. Full path name is provided for both source and include files.

Command Line

MICRO/C-51 has a built-in command-line processor that permits various options (switches) to control the compilation process. The format of the command line is as follows.

mcc51 filename.c [/switch...]

Optional switches

c – include C source in assembly source output file

dc – set default memory map (c = b,d,i,p,e)

f – enable function trace

ln – C source listing control (n = 1,2)

m – enable stack monitor

p – define processor descriptor file

wn – set warning report level (n = 0,1,2)

t – generate statement labels

Sample command line: mc51 test.c /de/l1/w0

Sample Program

The following MICRO/C-51 program initializes the 8051 serial-port and baud-rate-control registers and repeatedly calls on the C-library function "printf" to write the specified text string to the serial port.

```
main()
{
/* setup serial port (1200 BAUD @ 6 MHz) */
scon = 0x52;  /* set serial port control register */
tmod = 0x20;  /* set timer mode register          */
tcon = 0x69;  /* set timer control register       */
th1  = 0xf3;  /* set timer count                  */

while (1)  /* loop forever */
printf("hello, world\n");  /* write to serial port*/
}
```

Figure 10-3. Modular Programming Model

Modular Programs

MICRO/C-51 works with C-source modules (files) that contain either a complete program or part of a program. Individual program modules can be compiled or assembled separately to create relocatable object files. The Run-Time Library consists of a series of object modules organized into a library module. Once all the object modules are available, the linker/locator combines the object modules into a single executable program.

Assembler Relocation & Linkage Package

Assembler

* Gives symbolic access to powerful 8051 hardware features.

* Provides software support for many addressing and data allocation capabilities.

* Provides symbol table, cross-reference table, macro capabilities, and conditional assembly.

* Produces object files that can be linked together and located at absolute addresses.

Relocation & Linkage Package

* Links modules generated by the assembler and PL/M51.

* Locates linked object modules at absolute addresses.

* Creates libraries of object modules and has facilities for adding and deleting modules.

* Permits modules to be selectively linked from libraries.

* Converts 8051 objects into symbolic hexadecimal format to facilitate file-loading by symbolic hexadecimal loaders (such as non-Intel PROM programmers).

The Assembler Relocation & Linkage Package is a complete package for writing assembly-language programs to run on the powerful 8051 Family of microcontrollers. It includes the assembler, plus a relocation and linkage package that also contains a librarian, and an object-to-hex converter.

The assembler is a powerful assembly language that provides complete control over any microcontroller in the 8051 Family, enabling production of the most efficient code possible. With the assembler, the user can refer symbolically to many of the useful addressing features of the 8051. For example, symbolic references can be used for bit and byte locations, for 4-bit BCD arithmetic operations, for hardware registers, for I/O ports, for control bits, and for RAM addresses.

In addition, the assembler user can break up code into separately assembled modules, provide conditional-assembly capabilities, and support macros to automate frequently used code sequences.

The relocation and linkage package is used to prepare the program for running. The linker and relocator provides the facilities for combining program modules and assigning absolute addresses. The librarian gathers modules into a library where they can be accessed individually by the linker. The hex converter converts 8051 object modules into hexadecimal form in preparation for loading into ROM.

C Tools Tackle μC Software Bottlenecks

by Ed Thompson, Software Engineer
Micro Computer Control Inc., Hopewell NJ

Designing applications based on a single-chip microcomputer requires both hardware and software engineering skills. But the balance of these skills is changing as chip makers improve on-chip hardware capabilities at the expense of increased software complexity.

This change in the development environment has created a vigorous demand for alternatives to the time-honored assembly language coding method. Now, new development tools, based on C, promise to tackle this software bottleneck.

To the delight of many a hardware engineer, a wide variety of complex semiconductor devices is finding a welcome home on single-chip microcomputers. These include A/D converters, DMA controllers, intelligent communication receiver/transmitters, pulse-width modulators and event capture circuitry. This steadily growing engineers' wish list of on-chip resources is pushing single-chip microcomputer application in products that only a year ago would have required a boardful of chips.

Accelerating Demands

With this increased integration, however, comes a need for complex interfacing and control programming. Demands on software to control memory, interrupts and sophisticated peripheral devices are outpacing past design methods and leading to the adoption of programming methods once found solely in the realm of microprocessor-based designs.

A key area that is receiving a great deal of interest is the use of high-level languages for single-chip microcomputer program development. Although most of today's microprocessor-based projects use a high-level language as the primary coding language, this consideration has only recently been adopted on single-chip projects.

In the past, the limited size and complexity of the function to be coded and the lack of efficient high-level language compilers have restricted their consideration. But today the high cost of both software development and maintenance and the availability of efficient PC-based high-level language cross-compilers are quickly changing the way single-chip microcomputer programs are being developed.

Over the past five years, the C language, developed by AT&T for coding the Unix operating system, has gained an immense following in a broad range of applications. Whether C is best suited for all these diverse applications is another question.

However, since C was designed as an extensible, structured system-building language, it can support both high-level programming structures and low-level hardware interfaces. This capability, combined with some chip support extensions and an efficient code-generating implementation, make C worthy of consideration in single-chip microcomputer applications. One such implementation is Micro Computer Control's Micro/C-51 C compiler for the 8051 family of single-chip computers.

C comes with a long list of high-level statements and operators used to create structured programs that are quick to develop and easy to understand and maintain. But since C was not designed to cope with the special problems presented by single-chip microcomputers, a few well-chosen extensions are needed to make C a natural for this type of application. Three such extensions, implemented in Micro/C-51, are direct C support for memory maps, interrupts and access to on-chip peripherals.

The architecture of most single-chip microcomputers uses several memory maps. The 8051, being no exception, has no less than three on-chip and three off-chip memory maps. Memory size and access speed differ for each map. A problem arises in controlling the placement of variables in these various memory maps. One way to cope with this problem is to permit each declared C variable to be assigned to any map, thus providing easy adaptation to various target system memory configurations.

In addition to memory maps, interrupts also play an important role in most single-chip microcomputer applications. These interrupts are generated by internal or external peripheral devices, and indicate need for servicing by the processor. In some cases, up to a dozen or more interrupt sources must be serviced quickly. Support for interrupts could take the form of enabling a developer to assign any C function as the target of any interrupt source.

Access to peripheral devices presents a problem in any high-level language. Because single-chip microcomputers are used primarily in control applications, they especially demand a convenient and efficient access method to the increasing variety of on-chip peripherals. Here, a solution is to be able simply to use the name of a peripheral in a C expression to directly access the specified device. With such a simple yet powerful technique, even low-level device drivers become candidates for coding in C.

Software Debugging

With their integrated form of processor, memory and peripherals, single-chip microcomputers typically present a challenge to debugging efforts. Programming in a high-level language not only can reduce the entry of bugs in a program, but also can help in tracking them down.

The introduction of programming errors is reduced in several ways. Most reasonably, the fewer lines of high-level code needed to program a function simply reduces the chance of typographical errors that could go unnoticed. The procedural programming structure offered by C also helps in organizing the programming effort.

Nevertheless, the likelihood of creating a bug-free program is low. To help find the bugs, C debugging options can open the on-chip resources to inspection.

Single-chip microcomputers have proven to be an important product, and undoubtedly a host of new capabilities and architectures will soon emerge. As with microprocessors in the past, programming in a high-level language will help protect a company's investment in software when the time comes to exploit these new chips.

ARCHIMEDES C-8051 COMPILER

The Archimedes Microcontroller C-8051 Cross Compiler Kit supports software development for any chip based on the 8-bit 8051 microcontroller instruction set, e.g. 8051, 80C521, 80515 and other proliferation chips.

The C-8051 Kit consists of several pieces. The ANSI-standard C-compiler gives all the traditional high-level language advantages – faster coding, debugging and code maintenance resulting in more reliable code. The macroassembler is useful in optimizing any time-critical sections of code. It also preserves assembly code invest-ment by reassembling existing source code with the Archimedes assembler (which is linkable with C code). The assembler is highly compatible with other 8051 as-semblers. A librarian creates and maintains libraries. The linker combines C and assembly modules and places code and data at the right locations. The linker's numerous output formats make it quick and easy to support standard PROM programmers and emulators. (See Figure 10-4.)

Archimedes Microcontroller C-8051 is available on most popular software development hosts: IBM PC and com-patibles, MicroVAX and VAX systems running either VMS or UNIX (Ultrix or Berkeley). All versions are fully compat-ible, e.g. compile module 1 on a PC, module 2 on a MicroVAX/Ultrix system and link them on a VAX/VMS system.

Several Memory Models

The Archimedes Microcontroller C-8051 Kit has several memory models to best meet the requirements of different microcontroller designs, similar to 8086 small and large models. Memory models range from a small model using only the internal RAM (128/256 bytes) of an 8051 Family chip to a bankswitching model supporting up to 8 Mbytes of code. The different C-51 memory models are:

Small (single-chip) memory model: Supports 8051 configurations using internal RAM only. C variables and the run-time stack reside within internal RAM (128 or 256 bytes).

Medium (expanded) memory model: Supports microcontroller applications with a combined total of 64K code and data. Requires that the Program Status ENable signal ($\overline{PSEN}$) is AND-ed together with the Read Data signal ($\overline{RD}$), to create a uniform 64K address space. C variables and the run-time stack reside in external data memory.

Large (expanded) memory model: Supports micro-controller applications with 64K of code and 64K of data. C variables and the run-time stack reside in external data memory.

Banked memory model: Supports microcontroller applications with 64K of data and up to 8 Mbytes of code. C variables and the run-time stack reside in external data memory.

All memory models offers two approaches on how to allocate variables - reentrant or static. In the reentrant modes, all local "auto variables" are allocated and deallocated dynamically, i.e. they reside on a stack required to support recursive or reentrant functions. In the static modes , all function-level variables are forced into static memory with the exception of function argu-ments which are always on the stack.
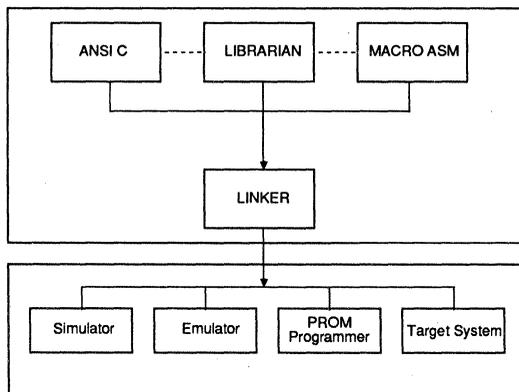
Figure 10-4.   C Kit

Table 10-3. Overview of Memory Models

Memory Model	Banked Reentrant	Banked Static	Expanded Reentrant	Expanded Static	Small Reentrant	Small Static
Typical chip	8031	8031	8031	8031	8051	8051
External RAM	Yes	Yes	Yes	Yes	No	No
Code Area	>1M	>1M	64K	64K	64K	64K
Recursion	Yes	No	Yes	No	Yes	No
C Interrupt Routines	Yes	Limited	Yes	Limited	Yes	Limited
C Variable Area	Ext. RAM (64K)	Ext. RAM (64K)	Ext. RAM (64K)	Ext. RAM (64K)	Int. RAM (256)	Int. RAM (256)
Relative Speed	Low	Low	Low	Medium	Medium	High
Relative Code Compactness	Medium	Medium	Medium	Medium	Medium	High

PROMable Code

PROMable code is a must for microcontroller applications. Archimedes supports PROMable code fully, including statically initialized data and static data without explicit initializers set to zero. The compiler has a simple invocation at compile time (-P) to automatically generate PROMable code.

C-Libraries

CHARACTER HANDLING <ctype.h>
isalnum, isalpha, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper, tolower, toupper

NON-LOGICAL JUMPS <setjmp.h>
longjmp, setjmp

FORMATTED INPUT/OUTPUT <stdio.h>
getchar, printf, putchar, sprintf, _formatted_write

GENERAL UTILITIES <stdlib.h>
exit, calloc, free, malloc, realloc

STRING HANDLING <string.h>
strcat, strcmp, strcpy, strlen, strncat, strncmp, strncpy

MATHEMATICS <math.h>
atan, atan2, cos, exp, log, log10, pow, sin, sqrt, tan

Archimedes C-8051 Compiler provides the most important C-library functions for stand alone "embedded microcontroller applications". "printf" can be used to make debugging easier or as the starting point for writing applications-specific display device drivers. Advanced math routines speed up number-intensive applications.

In addition, the C system contains C run-time libraries that are divided into 100+ small modules. By design, only those routines required by a particular program are called in at link-time to minimize run-time requirements (minimum 500 bytes; 2-3 kbytes for a typical application). All library routines are reentrant.

Fast Compilation

Single pass compilation, without any unnecessary assembly step, compiles 7000 lines of C source code in less than 30 seconds on a Compaq 386 system.

Fast Testing

ANSI-standard C makes it possible to compile "generic" C source code with different ANSI-standard C-compilers. Host-resident tools like Microsoft's C-86 compiler and CodeView debugger speed up testing of generic C-8051 code. (See Figure 10-5.)

ANSI-Standard Power and Features

ANSI-standard C has some extra features over and above the traditional K&R C language definition. Function prototyping allows function declarations a la Pascal with the conversion conventions of C. This speeds up software development and produces more efficient code, by avoiding some of the default conversions to "int" that is typically required in older C compilers. "Structure" and "union" assign and "enum" types give the same facilities enjoyed by Pascal users. Flexible "auto" initialized aggregates like arrays, structures and unions provide one more option to keep vital data local to a function rather than making everything global.
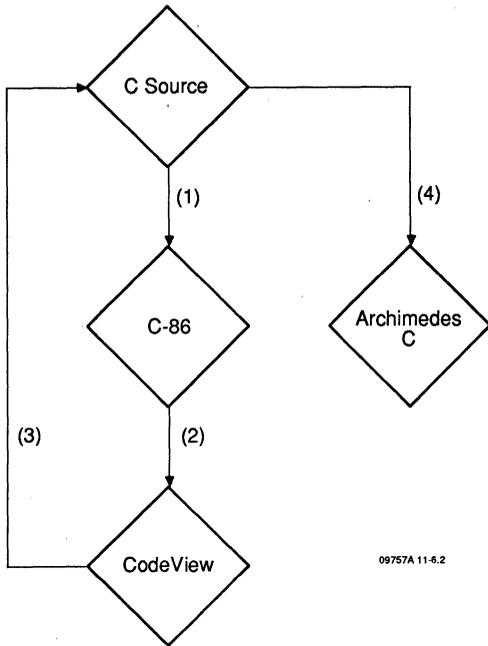
Figure 10-5. Testing using Host Tools
like Microsoft C-86

All the Standard ANSI Data Types

Archimedes C-8051 compiler supports all the basic ANSI C elements. Object sizes in bytes:

char	short	int	long	float	pointer/addr
1	1/2	2	4	4	2

"Float" is implemented in the IEEE 32-bit single-precision format.

LINT-type Feature

The software has a built-in advanced type-checking scheme to eliminate difficult to find "typing errors" and to speed up integrating different modules. The C compiler checks a module, whereas the linker checks consistency of inter-module declarations (down to the last bit of a complex structure). This facilitates interfacing of libraries, or other routines only available in object format, as well as integrating modules written by different programmers.

Error Message System

To speed up error searching the C compiler has a state-of-the-art built-in error message system (invoked by the -V switch). The system indicates the exact source code location and a message describes the error detected:

```
if(i))j++
_____^
```
"main.c", 870 Error (110): ')' unexpected

C Language Extensions and Other Specials

The Archimedes C-8051 kit has special C language extensions, or built-in in-line functions, to better take advantage of a chip's special features and speed up development. "input" and "output" provide access to internal RAM/special function registers. Functions like "set bit" and "clear bit" are available to support bit manipulation. Also, functions are available to read blocks of code and data.

The C compiler has several special listing options. It can for instance generate a pure assembly source file (-A option), which can be hand-optimized and then reassembled with the macroassembler. A list file with mixed C source and native assembly code speeds up debugging. The C compiler supports symbols with up to 255 significant characters.

Linker

The linker combines C and assembly modules and automatically links in the necessary C run-time libraries (including the C start-up routine). The flexible linker locates memory segments at absolute or relocatable addresses. The linker's many output options provide fast and easy interfacing with most PROM-programmers and emulators. The Archimedes kit generates symbolic debug information for global and local static variables as well as line numbers. The linker also generates load maps and module/symbol cross-reference listings to make debugging faster.

C — the Right Choice for the Right Project

Why spend months of extra development time to save some money on memory chips? Constantly lower memory prices have reduced the need to save on every byte of memory. Typically, only in high-volume applications, do the cost savings in memory chips from assembly programming justify the extra costs in development time. (See Figure 10-6.)

In low and medium volume applications, C is the right choice. Development time and costs are cut by at least 50% and the product goes out the door faster–all for minimal extra memory costs per system.

C is also the right choice for projects on a tight time schedule and for any products requiring complex software development. Assembly programming might be best if most code is very time-critical.

Archimedes Microcontroller C-8051 Kit comes with both a C compiler and a macroassembler to provide optimal flexibility. C speeds up software development and the macro assembler can be used to optimize time-critical sections of code, where necessary.
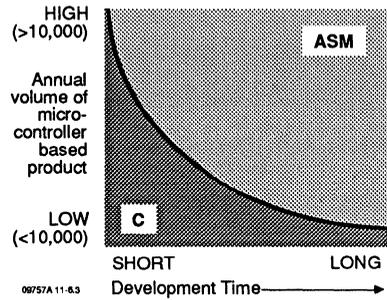
Figure 10-6.   C - Right Tool for the Right Project

DATA I/O PROGRAMMERS

How Programmers Work

Programmers apply very specific voltages to device pins to "blow" a fuse and thereby record a value, either memory or logical. Programming waveforms are generated from raw programmer power supplies using regulators controlled by the programmer's microprocessor. The specific power, rise and fall, etc. of the charge are specified by device-specific algorithms recorded within the programmer.

Values for programming variables, including pinouts, voltage levels and timing, are stored in firmware or floppy tables. When a particular device is chosen, the programmer uses information stored in these tables to assemble a device-specific programming routine in scratch RAM. Device pinout variations are handled by different device sockets, cartridges or modules on the programmer or pak. Newer programmers such as Data I/O's UniSite can program any device up to 40 pins on one socket. To maximize control speed during programming, the programmer and pak make extensive use of addressable latches for control signals.

Programmers range in price from under $500 to over $15,000. Along with basic capability, part of the price differential is the result of more established programmer manufacturers establishing a system of seeking semiconductor manufacturers' approvals for device support. Data I/O works closely with the device makers to support a new device before silicon is available. When samples are available the device maker approves device support.

Programmer Controls

Data I/O programmers can obtain data from three sources; a master device, a serial port/disk drive, or from the keyboard. Master devices are first copied into the programmer RAM where the code can be edited at the bit level or copied onto other media. Code can be edited using the integral keyboard or by loading it into a PC and editing it onscreen. On most Data I/O programmers, a standard terminal will also enable the code to be edited on screen.

PROMlink is Data I/O's optional PC-based control software for all of Data I/O's programmers. It enables the user to control any programmer from a simple menu system, storing data and configuration files on hard or floppy disk. It allows simple bit-editing functions in ASCII or Hex and will convert from one to the other. It also has a simple device labeling function using a standard PC printer.

Programmers can be networked and assigned a node identification on most workstation networks, such as PC, UNIX or VAX. This allows centralized device data storage for both engineering and testing groups and facilitates data transfer. An engineer can develop a design at a PC or workstation node and download to a remote programmer.

Device files are generally kept on disk or on master devices. Programmers require updates to be able to program the most current devices and these updates are also provided on firmware, i.e. programmed devices, or floppy disks. Data I/O offers annual update services which automatically keep a programmer at the most current revision.

All data transfer or verification operations take place between the programmer's internal RAM and the device or between the RAM and serial port or floppy drive in the programmer. Because the operation procedure to transfer data via a serial port varies from programmer to programmer, we will describe data transfer with the most widely used system. All of these functions can occur from the programmer front panel or from a remote terminal.

Typical Programmer Operation Steps

- Load RAM with data from a master device.
- Press COPY and the programmer will prompt COPY DATA FROM.
- Select DEVICE and the programmer prompts DEV ^ADDR/SIZE TO.
- Select RAM and the programmer prompts CO DEV>RAM ^ ADDR.
- Press START and the programmer will lead through the device selection process to identify master device type.
- Place the master device in the main programmer socket and press START to load data into RAM. From RAM it can be programmed into a device different from the master or stored on floppy disk.
- Verify RAM against the master device.
- Program a new device with RAM data.

Data editing is possible while data is in RAM. The programmer allows simple bit editing on the internal LED command line screen or on a remote terminal or PC. Using PROMlink for full screen editing on a PC allows editing/input in ASCII or Hex and automatic conversion from one to the other.

Set programming allows the downloading of an entire data file into RAM (up to a maximum of 512K bytes on most programmers with 1 Mbyte coming soon) in one operation. The data is automatically split according to word width into as many devices as required, which are then programmed sequentially.

Programmer Types and Technology

An "Engineering" programmer is generally a stand-alone one-device-at-a-time programmer. Models are available that do memory only, logic only or memory and logic.

Inexpensive memory-only programmers are often appropriate for the first-time user. They are usually in the $1,000 range for a name brand and generally support MOS/CMOS EPROMs and EEPROMs up to 512K bits. The better ones support 8-,16- and 32-bit-wide words and may be run from the front panel keyboard or by an optional PC interface.

Universal logic and memory programmers are the "workhorse" engineering programmers. Most engineers prefer them for their flexibility and adaptability to future device needs. They generally consist of a mainframe unit containing the power supply, primary microprocessor, memory, keypad and control functions. Modules or paks are then added to characterize the mainframe for memory, multiple memory or logic (see page 10-26). The most popular units translate data from 29 or more popular formats and have up to 1 Mbyte of internal RAM.

Functional Specifications for the Data I/O 29B System:

- General Architecture: Microprocessor controlled
- Data RAM: 256 x 8 standard, upgrades available to 1 Mbyte
- Programming Support: GangPak, LogicPak, UniPak 2B, MOSPak, and programming modules
- Keyboard: 16-key hexadecimal and 9-key functional
- Functional keys:
  Copy: Used to move a block of data to or from a serial port, RAM, or device. Works in conjunction with source/destination keys.
  Verify: Used to make a byte-by-byte comparison of a block of data. Used with source/destination keys.
  Select: Prepares the programmer to accept codes for select functions.
  Edit: Allows viewing and changing of data at individually selected RAM address locations

- Display: 16-character alphanumeric
- Input/Output: Serial RS-232C and 20mA current loop
- Baud Rates: 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600
- Remote Control: PROMlink (MS-DOS) optional
  Computer Remote Control (CRC)
  Terminal Remote Control (SRC)
- Translation Formats: 29 available
- Handler Capability: Optional handler port is available for binning and control signals

Pin-driver technology programmers are the newest programming technology. They use a dedicated voltage driver for each pin, enabling each programming socket pin to be configured by software to execute device-specific information including voltage, current, logic level, ground and Vcc outputs.

Gang programmers or gang programming paks have a master socket and usually seven slave sockets. They are useful in the engineering environment or limited production runs, to run small batches of identical parts or to do set programming. In the set-programming mode, most gang programmers allow several sets to be programmed at once.

Production programmers are high-throughput programming and test fixtures intended for the production floor. For devices that program rapidly, the most common method is serial programming, whereby a single-socket programmer is connected to an automatic device handler that runs chips individually by a programming/test head. Most memory devices program most efficiently on a parallel programmer whereby 10 or 20 devices are loaded into individual programming/test sockets and are programmed at once. More recently designed models such as Data I/O's Series 1000 have "rails" whereby the device sockets are aligned end to end and entire tubes can be smoothly loaded, programmed and unloaded. Sophisticated production programmers such as Data I/O's Series 1000 can also serialize devices in specified areas of device memory, label devices and provide simple code-editing capability.

The programming pass also includes tests for continuity, incorrectly inserted devices and a data comparison with RAM. On programmers like the Series 1000, full programming pass/fail statistical data is accumulated by time of day, socket and device. Calibration is automatic and production statistics can be stored on disk.

In-circuit programming entails programming a device or devices already mounted on a board. The programmable devices are soldered in place and programmed through a specially designed edge-connector. Boards must be designed from the beginning to accommodate the technology and to protect microprocessors from higher voltages. For certain types of applications the additional effort can be worth it. Typical reasons for adopting an in-circuit design include the elimination of additional device handling and increased board reliability. Specific reasons for avoiding individual device programming include the following:

- High device count per board statistically increases the chance of physically damaged devices during handling.

- Frequently updated code, requiring excess removal, downtime or additional boards to control the board float.

- Surface-mount devices particularly defy modifications if they are not in-circuit programmable.

- Soldered-in designs, especially military design specs which often require soldered-in devices, are difficult or impossible to remove.

Data I/O supports the full line of programmable products from AMD including the 87C51, 87C52T2, 87C521, 87C541, 8751H and 8753H microcontrollers.

Programmer Systems Overviews

Unisite 40 supports every microcontroller, PROM, EPROM, EEPROM, PLD, IFL and FPLA that fits in its 40-pin DIP socket. The optional ChipSite module adds a single site for PLCCs, LCCs and SOICs. Unisite 40 uses universal pin drivers to drive each pin to any state needed to program and test a programmable device. The system provides 128K bytes of RAM and two disk drives as standard; 1 Mbyte of internal memory is available on order. Updates are provided on 3 1/2" floppy disks.
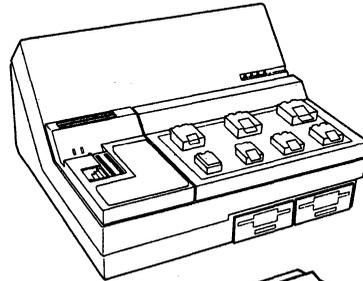
The 29B System provides a universal system for programming, testing and verifying a variety of memory and logic devices. The 29B can be tailored to specific programming needs by selecting the appropriate programming pak, shown below, and simply plugging it into the 29B.
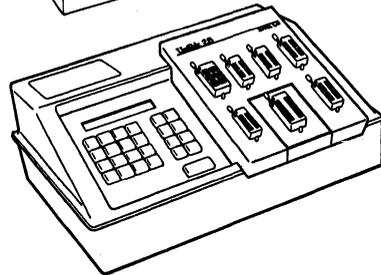
Programming Paks

- **Unipak 2B** programs more than 1200 devices, including MOS and CMOS EPROMs and EEPROMs, fuse link, AIM and DEAP bipolar PROMs. Simple pinout cartridges are available for 40-pin microcomputers and parts with non-standard pinouts and unique package types (LCC, PLCC).

- **LogicPak** combined with appropriate plug-in adapters, allows you to design, program and functionally test more than 440 different logic devices.
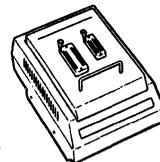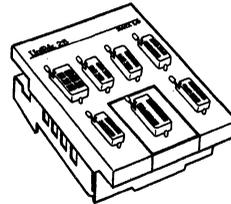
CHAPTER 11

Package Outlines

Package Outlines

PHYSICAL DIMENSIONS*

Plastic Dual-In-Line Package (PD)
PD 040

2.040
2.080

.530
.580

.005
MIN.

.045
.065

.090
.110

.015
.060

.140
.225

.125
.160

.014
.023

.580
.620

.008
.015

0°
15°
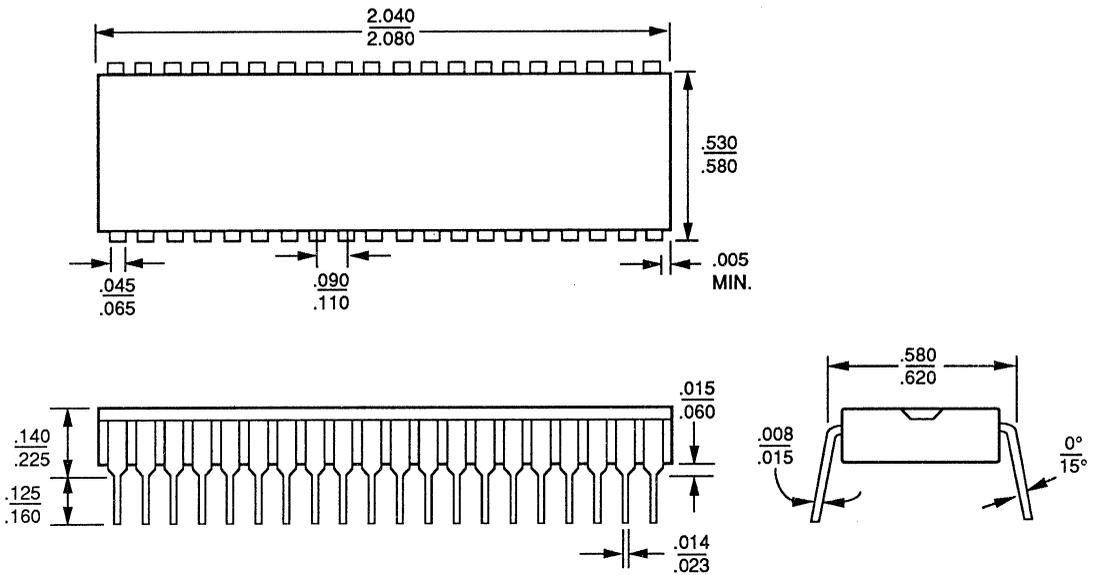
PID# 06823B

* For reference only.

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Ceramic Hermetic Dual-In-Line Packages (CD/CDV)

CD 040

2.035
2.080

.098
MAX

.565
.605

1

.050
.065

.100
BSC

.005
MIN

.015
.060

.160
.220

.125
.160

.015
.022

.590
.615

0°
15°

.008
.012

.150
MIN

.700
MAX

06824C

CDV 040

2.035
2.080

.098
MAX

.565
.605

1

.050
.065

.100
BSC

.005
MIN

.015
.060

.160
.220

.125
.160

.015
.022

.590
.615

0°
15°

.008
.012

.150
MIN

.700
MAX
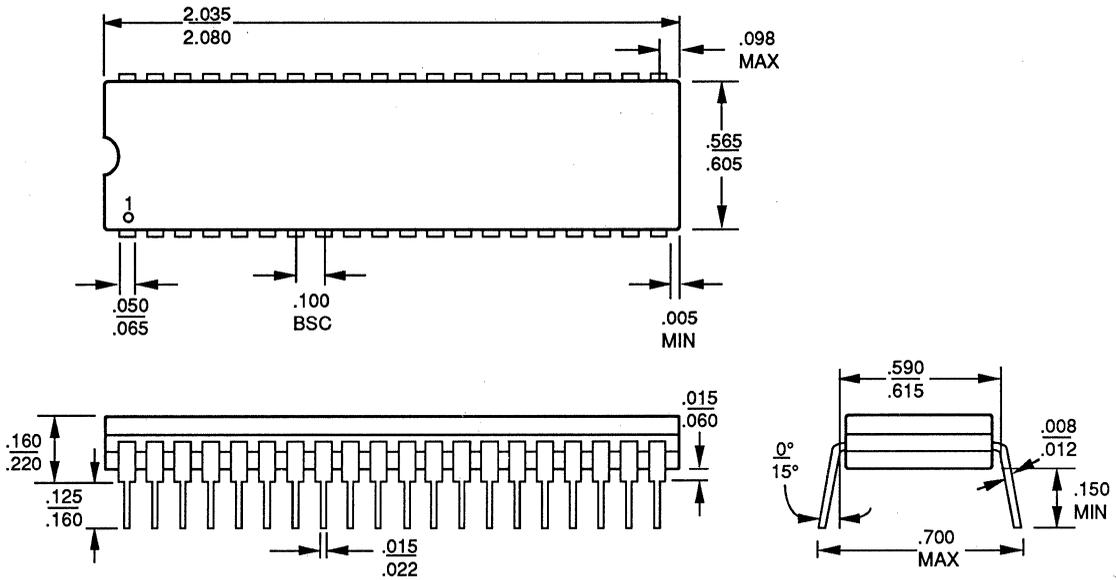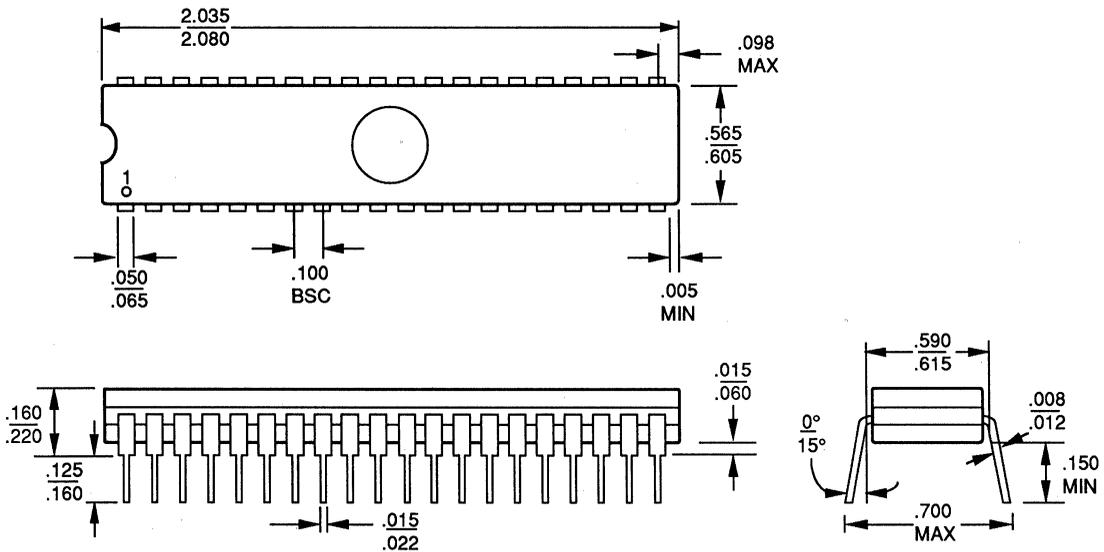
07880C

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Plastic Leaded Chip Carriers (PL)
PL 044

.042
.048

.050
REF.

.042
.056

.009
.015

.025
.045

1

O

.045
TYP.

.685
.695

.650
.656

.590
.630

.026
.032

.500
REF.

.013
.021

.650
.656

.685
.695

.090
.120

.009
.015

.165
.175

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

11-3

Ceramic Leadless Chip Carrier (CLV)
CLV 044
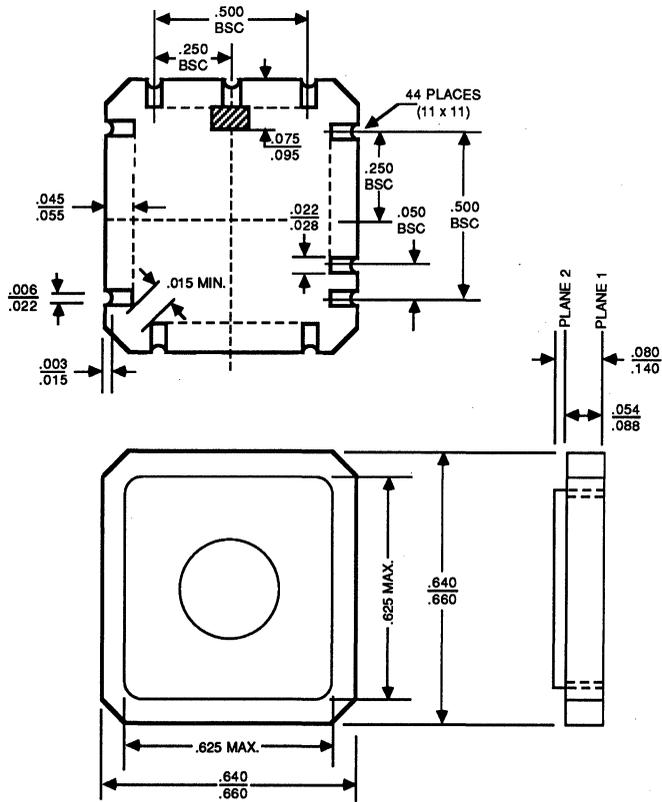
PID #09703C

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Notes