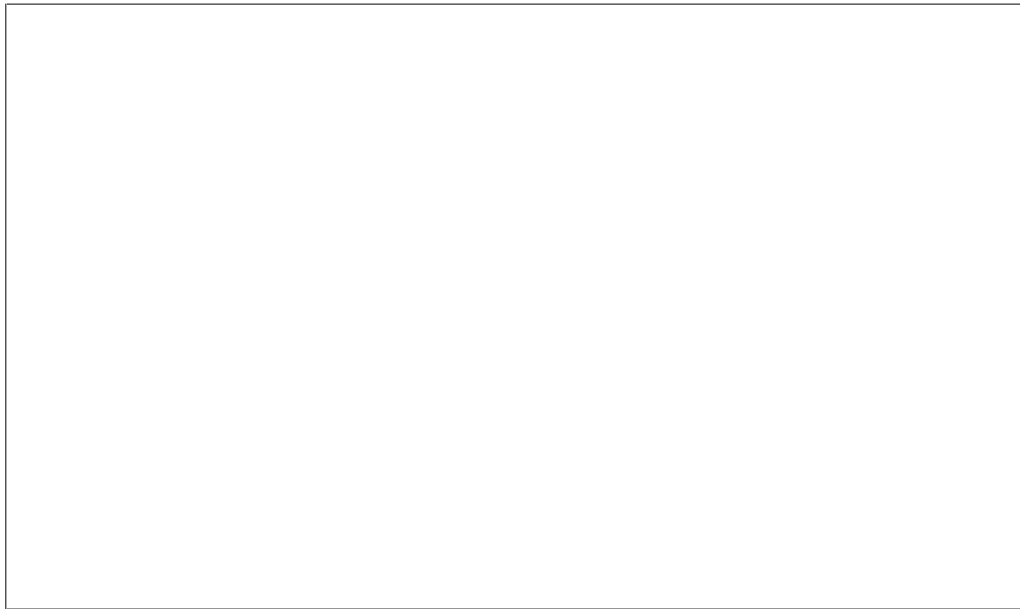


SIEMENS



Microcomputer Components

SAB 80515 / SAB 80C515

SAB 80C515 / SAB 80C535

SAB 80515 / SAB 80535

8-Bit Single-Chip Microcontroller Family

User's Manual 08.95

SAB 80515 / SAB 80C515 Family	
Revision History: 8.95	
Previous Releases: 12.90/10.92	
Page	Subjects (changes since last revision)
30	Modified timing diagram ($\overline{\text{PSEN}}$ rising edge)
39	More detailed description of ACMOS port structure
80	Differential output impedance of analog reference supply voltage now: 1 k Ω
105	Second paragraph: additional description; WDT reset information added
106	SWDT reset information added
109	Figure 7-51 corrected
137	Encoding of ADD A, direct corrected
152	Encoding of CPL bit corrected
243	New release of SAB 80C515 / SAB 80C535 data sheet inserted
301	New release of SAB 80515 / SAB 80535 data sheet inserted

Edition 08.95

**Published by Siemens AG,
Bereich Halbleiter, Marketing-
Kommunikation, Balanstraße 73,
81541 München**

© Siemens AG 1995.
All Rights Reserved.

Attention please!

As far as patents or other rights of third parties are concerned, liability is only assumed for components, not for applications, processes and circuits implemented within components or assemblies.

The information describes the type of component and shall not be considered as assured characteristics.

Terms of delivery and rights to change design reserved.

For questions on technology, delivery and prices please contact the Semiconductor Group Offices in Germany or the Siemens Companies and Representatives worldwide (see address list).

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Siemens Office, Semiconductor Group.

Siemens AG is an approved CECC manufacturer.

Packing

Please use the recycling operators known to you. We can also help you – get in touch with your nearest sales office. By agreement we will take packing material back, if it is sorted. You must bear the costs of transport.

For packing material that is returned to us unsorted or which we are not obliged to accept, we shall have to invoice you for any costs incurred.

Components used in life-support devices or systems must be expressly authorized for such purpose!

Critical components¹ of the Semiconductor Group of Siemens AG, may only be used in life-support devices or systems² with the express written approval of the Semiconductor Group of Siemens AG.

- 1 A critical component is a component used in a life-support device or system whose failure can reasonably be expected to cause the failure of that life-support device or system, or to affect its safety or effectiveness of that device or system.
- 2 Life support devices or systems are intended (a) to be implanted in the human body, or (b) to support and/or maintain and sustain human life. If they fail, it is reasonable to assume that the health of the user may be endangered.

Contents	Page
1 Introduction	6
2 Fundamental Structure	10
2.1 Differences between MYMOS (SAB 80515/80535) and ACMOS (SAB 80C515/80C535) Versions	13
2.1.1 Power Saving Modes	13
2.1.2 Special Function Register PCON	13
2.1.3 Port Driver Circuitries	14
2.1.4 The A/D Converter Input Ports	14
2.1.5 A/D Converter Timings	15
2.1.6 The Oscillator and Clock Circuits	15
2.1.7 The V_{BB} Pin	15
3 Central Processing Unit	16
3.1 General Description	16
3.2 CPU Timing	17
4 Memory Organization	19
4.1 Program Memory	19
4.2 Data Memory	19
4.3 General Purpose Register	23
4.4 Special Function Registers	23
5 External Bus Interface	27
5.1 Accessing External Memory	27
5.2 \overline{PSEN} , Program Store Enable	29
5.3 ALE, Address Latch Enable	29
5.4 Overlapping External Data and Program Memory Spaces	29
6 System Reset	31
6.1 Hardware Reset and Power-Up Reset	31
6.1.1 Reset Function and Circuitries	31
6.1.2 Hardware Reset Timing	34
7 On-Chip Peripheral Components	35
7.1 Parallel I/O	35
7.1.1 Port Structures	35
7.1.1.1 Digital I/O Port Circuitry (MYMOS/ACMOS)	36
7.1.1.2 MYMOS Port Driver Circuitry	39
7.1.1.3 ACMOS Port Driver Circuitry	39
7.1.2 Port 0 and Port 2 Used as Address/Data Bus	41
7.1.3 Alternate Functions	42
7.1.4 Port Handling	44
7.1.4.1 Port Timing	44

Contents (cont'd)	Page
7.1.4.2 Port Loading and Interfacing	45
7.1.4.3 Read-Modify-Write Feature of Ports 0 through 5	45
7.2 Serial Interfaces	47
7.2.1 Operating Modes of Serial Interface	47
7.2.2 Multiprocessor Communication Feature	50
7.2.3 Baud Rates	50
7.2.4 Detailed Description of the Operating Modes	54
7.2.4.1 Mode 0, Synchronous Mode	54
7.2.4.2 Mode 1, 8-Bit UART	55
7.2.4.3 Mode 2, 9-Bit UART	57
7.2.4.4 Mode 3, 9-Bit UART	57
7.3 Timer 0 and Timer 1	65
7.3.1 Mode 0	68
7.3.2 Mode 1	69
7.3.3 Mode 2	70
7.3.4 Mode 3	71
7.4 A/D Converter	72
7.4.1 Function and Control	74
7.4.1.1 Initialization and Input Channel Selection	74
7.4.1.2 Start of Conversion	75
7.4.2 Reference Voltages	76
7.4.3 A/D Converter Timing	80
7.5 Timer 2 with Additional Compare/Capture/Reload	82
7.5.1 Timer 2	85
7.5.2 Compare Function of Registers CRC, CC1 to CC3	88
7.5.2.1 Compare Mode 0	88
7.5.2.2 Compare Mode 1	92
7.5.2.3 Using Interrupts in Combination with the Compare Function	94
7.5.3 Capture Function	96
7.6 Power Saving Modes	98
7.6.1 Power Saving Modes of the SAB 80515/80535	99
7.6.1.1 Power-Down Mode of the SAB 80515/80535	99
7.6.2 Power Saving Modes of the SAB 80515/80535	100
7.6.2.1 Power-Down Mode of the SAB 80C515/80C535	101
7.6.2.2 Idle Mode of the SAB 80C515/80C535	102
7.7 Watchdog Timer	105
7.8 Oscillator and Clock Circuit	107
7.8.1 Crystal Oscillator Mode	107
7.8.2 Driving for External Source	107
7.8.2.1 Driving the SAB 80515/80535 from External Source	108
7.8.2.2 Driving the SAB 80C515/80C535 from External Source	109

Contents (cont'd)	Page
7.9 System Clock Output	110
8 Interrupt System	112
8.1 Interrupt Structure	112
8.2 Priority Level Structure	120
8.3 How Interrupts are Handled	122
8.4 External Interrupts	125
8.5 Response Time	126
9 Instruction Set	127
9.1 Addressing Modes	127
9.2 Introduction to the Instruction Set	129
9.2.1 Data Transfer	129
9.2.2 Arithmetic	130
9.2.3 Logic	132
9.2.4 Control Transfer	132
9.3 Instruction Definitions	134
10 Device Specifications	214
Data Sheet SAB 80C515/80C535	
Data Sheet SAB 80515/80535	

1 Introduction

The SAB 80C515/80C535 is a new, powerful member of the Siemens SAB 8051 family of 8-bit microcontrollers. It is designed in Siemens ACMOS technology and is functionally compatible with the SAB 80515/80535 devices designed in MYMOS technology.

The ACMOS and the MYMOS versions ^{1) 2)} are stand-alone, high-performance single-chip microcontrollers based on the SAB 8051/80C51 architecture. While maintaining all the SAB 80(C)51 operating characteristics, the SAB 80(C)515/80(C)535³⁾ incorporate several enhancements which significantly increase design flexibility and overall system performance.

The low-power properties of Siemens ACMOS technology allow applications where power consumption and dissipation are critical. Furthermore, the SAB 80C515/80C535 has two software-selectable modes of reduced activity for further power reduction: idle and power-down mode.

The SAB 80(C)535 is identical to the SAB 80(C)515 except that it lacks the on-chip program memory. The SAB 80(C)515/80(C)535 is supplied in a 68-pin plastic leaded chip carrier package (P-LCC-68). In addition to the standard temperature range version (0 ° to + 70 °C) there are also versions for extended temperature ranges available (see data sheets).

Functional Description

The members of the SAB 80515 family of microcontrollers are:

- SAB 80C515: Microcontroller, designed in Siemens ACMOS technology, with 8-Kbyte factory mask-programmable ROM
- SAB 80C535: ROM-less version, identical to the SAB 80C515
- SAB 80515: Microcontroller, designed in Siemens MYMOS technology, with 8-Kbyte factory mask-programmable ROM
- SAB 80535: ROM-less version, identical to the SAB 80515
- SAB 80515K: Special ROM-less version of the SAB 80515 with an additional interface for program memory accesses. An external ROM that is accessed via the interface substitutes the SAB 80515's internal ROM.

1 In this User's Manual the term "ACMOS versions" is used to refer to both the SAB 80C515 and SAB 80C535.

2 The term "MYMOS versions" stands for SAB 80535 and SAB 80515.

3 The term "SAB 80(C)515" refers to the SAB 80515 and the SAB 80C515, unless otherwise noted.

The SAB 80(C)515 features are:

- 8 Kbyte on-chip program memory
- 256 byte on-chip RAM
- Six 8-bit parallel I/O ports
- One input port for digital input ¹⁾
- Full-duplex serial port, 4 modes of operation, fixed or variable baud rates
- Three 16-bit timer/counters
- 16-bit reload, compare, capture capability
- A/D converter, 8 multiplexed analog inputs, programmable reference voltages
- 16-bit watchdog timer
- Power-down supply for 40 byte of RAM
- Boolean processor
- 256 directly addressable bits
- 12 interrupt sources (7 external, 5 internal), 4 priority levels
- Stack depth up to 256 byte
- 1 μ s instruction cycle at 12-MHz operation
- 4 μ s multiply and divide
- External program and data memory expandable up to 64 Kbyte each
- Compatible with standard SAB 8080/8085 peripherals and memories
- Space-saving P-LCC-68 package

For small-quantity applications and system development the SAB 80535 can be employed being the equivalent of an SAB 80515 without on-chip ROM.

1 Additional feature of the AC MOS versions

Figure 1-1 shows the logic symbol, figure 1-2 the block diagram of the SAB 80(C)515:

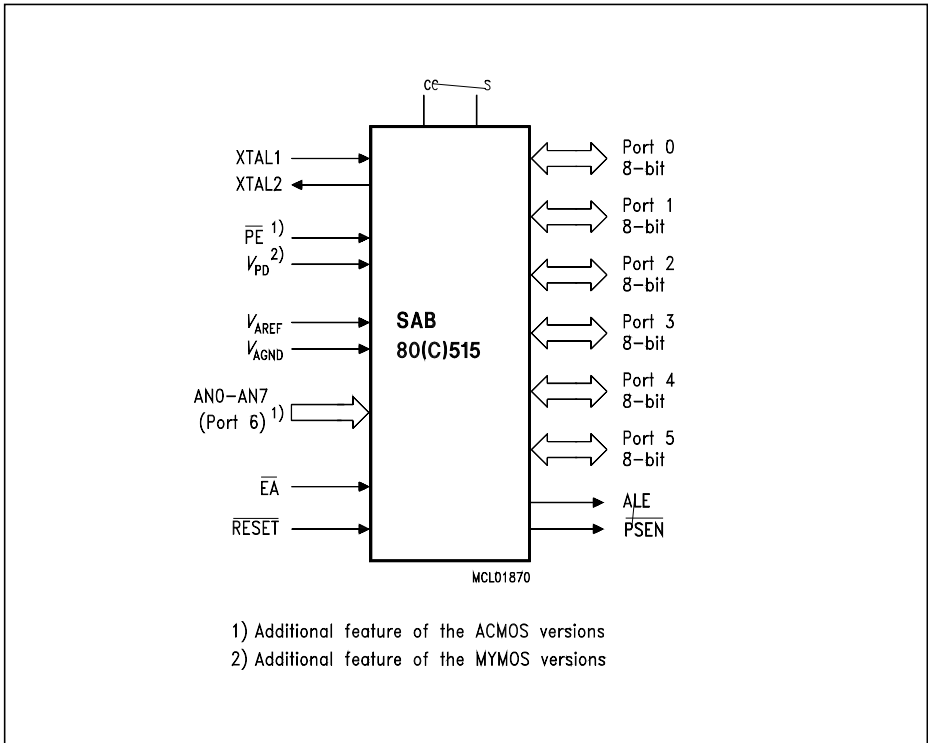


Figure 1-1
Logic Symbol

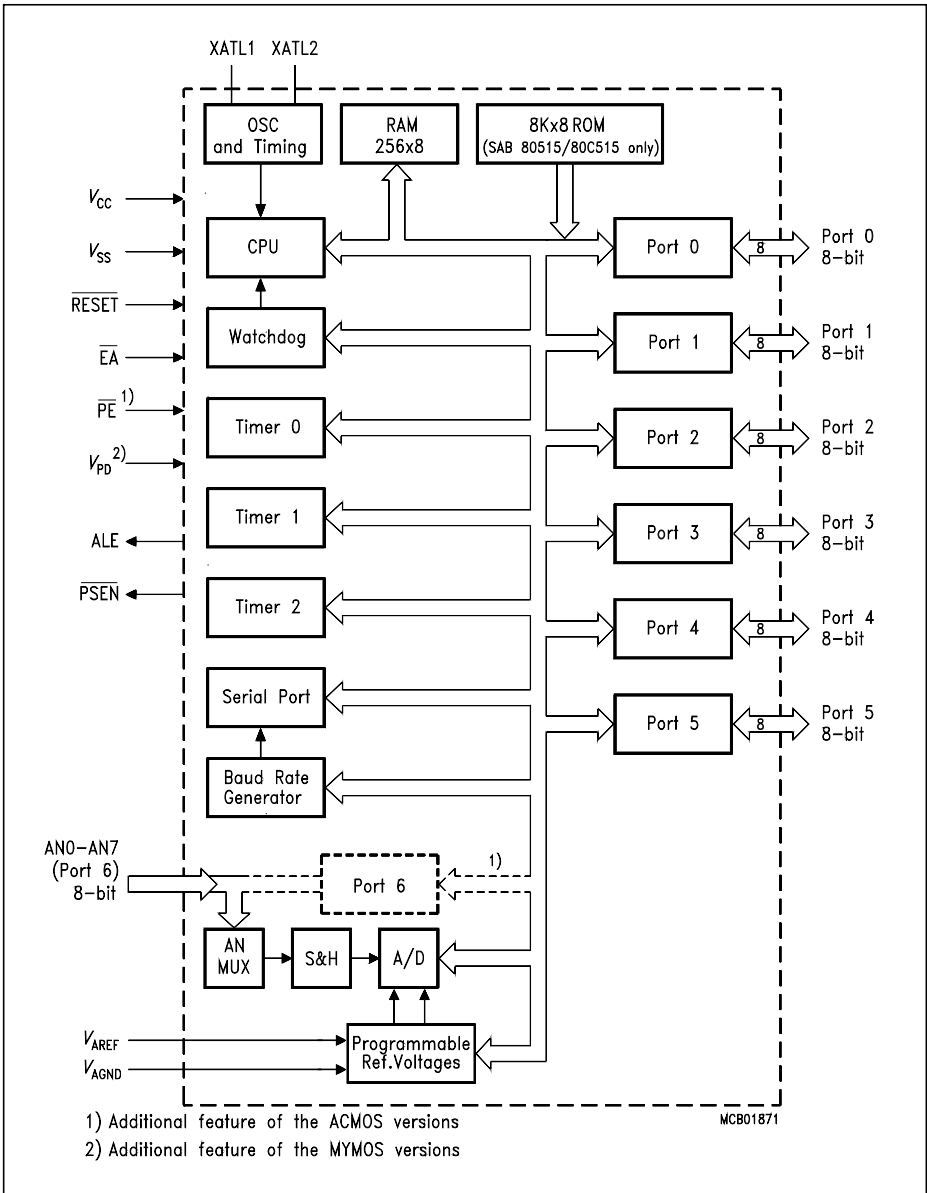


Figure 1-2
 Block Diagram

2 Fundamental Structure

The SAB 80(C)515/80(C)535 is a totally 8051-compatible microcontroller while its peripheral performance has been increased significantly.

Some of the various peripherals have been added to support the 8-bit core in case of stringent embedded control requirements without losing compatibility to the 8051 architecture.

Furthermore, the SAB 80(C)515/80(C)535 contains e. g. an additional 8-bit A/D converter, two times as much ROM and RAM as the 80(C)51 and an additional timer with compare/capture/reload facilities for all kinds of digital signal processing.

Figure 2.1 shows a block diagram of the SAB 80(C)515/80(C)535.

The SAB 80C515/80C535 combines the powerful architecture of the industry standard controller SAB 80515/80535 with the advantages of the ACMOS technology (e. g. power-saving modes). The differences between MYMOS and ACMOS components are explained in section 2.1.

Readers who are familiar with the SAB 8051 may concentrate on chapters 2.1, 6, 7 and 8 where the differences between MYMOS and ACMOS components, the reset conditions, the peripherals and the interrupt system are described.

For newcomers to the 8051 family of microcontrollers, the following section gives a general view of the basic characteristics of the SAB 80515/80535. The details of operation are described later in chapters 3 and 4.

Central Processing Unit

The CPU is designed to operate on bits and bytes. The instructions, which consist of up to 3 bytes, are performed in one, two or four machine cycles. One machine cycle requires twelve oscillator cycles. The instruction set has extensive facilities for data transfer, logic and arithmetic instructions. The Boolean processor has its own full-featured and bit-based instructions within the instruction set. The SAB 80(C)515/80(C)535 uses five addressing modes: direct access, immediate, register, register indirect access, and for accessing the external data or program memory portions a base register plus index-register indirect addressing.

Memory Organization

The SAB 80C515, 80515 have an internal ROM of 8 Kbyte. The program memory can externally be expanded up to 64 Kbyte (see bus expansion control). The internal RAM consists of 256 bytes. Within this address space there are 128 bit-addressable locations and four register banks, each with 8 general purpose registers. In addition to the internal RAM there is a further 128-byte address space for the special function registers, which are described in sections to follow.

Because of its Harvard architecture, the SAB 80(C)515/80(C)535 distinguishes between an external program memory portion (as mentioned above) and up to 64 Kbyte external data memory accessed by a set of special instructions.

Bus Expansion Control

The external bus interface of the SAB 80(C)515/80(C)535 consists of an 8-bit data bus (port 0), a 16-bit address bus (port 0 and port 2) and five control lines. The address latch enable signal (ALE) is used to demultiplex address and data of port 0. The program memory is accessed by the program store enable signal (\overline{PSEN}) twice a machine cycle. A separate external access line (\overline{EA}) is used to inform the controller while executing out of the lower 8 Kbyte of the program memory, whether to operate out of the internal or external program memory. The read or write strobe (\overline{RD} , \overline{WR}) is used for accessing the external data memory.

Peripheral Control

All on-chip peripheral components - I/O ports, serial interface, timers, compare/capture registers, the interrupt controller and the A/D converter - are handled and controlled by the so-called special function registers. These registers constitute the easy-to-handle interface with the peripherals. This peripheral control concept, as implemented in the SAB 8051, provides the high flexibility for further expansion as done in the SAB 80(C)515/80(C)535.

Moreover some of the special function registers, like accumulator, B-register, program status word (PSW), stack pointer (SP) and the data pointer (DPTR) are used by the CPU and maintain the machine status.

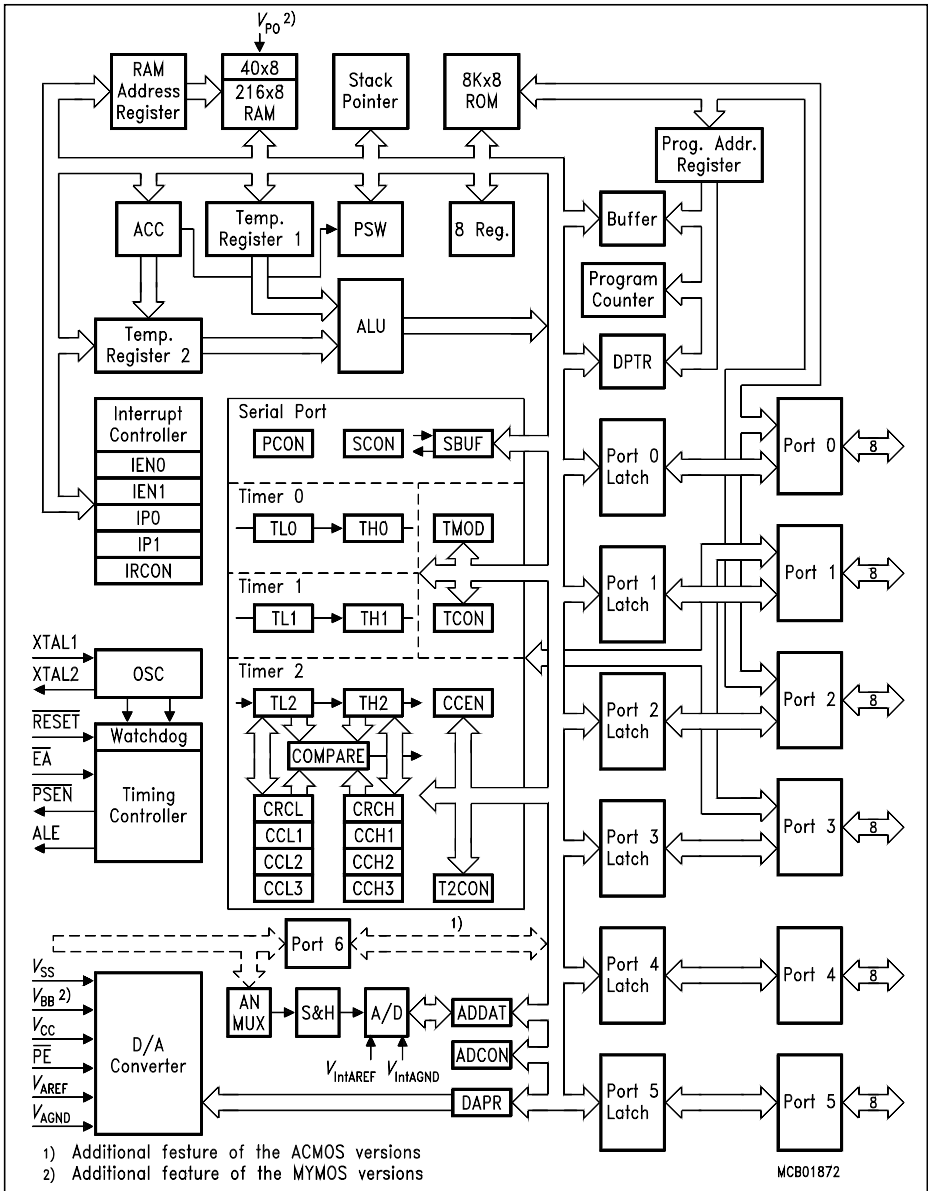


Figure 2-1
Detailed Block Diagram

2.1 Differences between MYMOS (SAB 80515/80535) and ACMOS (SAB 80C515/80C535) Versions

There are some differences between MYMOS and ACMOS versions concerning:

- Power Saving Modes
- Special Function Register PCON
- Port Driver Circuitry
- A/D Converter Input Ports
- A/D Converter Conversion Time
- Oscillator and Clock Circuit
- V_{BB} Pin

2.1.1 Power Saving Modes

The SAB 80515/80535 has just the power-down mode, which allows retention of the on-chip RAM contents through a backup supply connected to the V_{PD} pin.

The SAB 80C515/80C535 additionally has the following features:

- idle mode
- the same power supply pin V_{CC} for active, power-down and idle mode
- an extra pin (\overline{PE}) that allows enabling/disabling the power saving modes
- starting of the power-saving modes by software via special function register PCON (Power Control Register)
- protection against unintentional starting of the power-saving modes

These items are described in detail in section 7.6.

2.1.2 Special Function Register PCON

In the MYMOS version SAB 80515/80535 the SFR PCON (address 87_H) contains only bit 7 (SMOD).

In the ACMOS version SAB 80C515/80C535 there are additional bits used (**see figure 2-2**).

The bits PDE, PDS and IDLE, IDLS select the power-down mode or idle mode, respectively, when the power saving modes are enabled by pin \overline{PE} .

Furthermore, register PCON of the ACMOS version contains two general-purpose flags. For example, the flag bits GF0 and GF1 can be used to indicate whether an interrupt has occurred during normal operation or during idle. Then 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 sample the flag bits.

2.1.3 Port Driver Circuitries

The port structures of the MYMOS and ACMOS versions are functionally compatible. For low power consumption the pullup arrangement is realized differently in both versions.

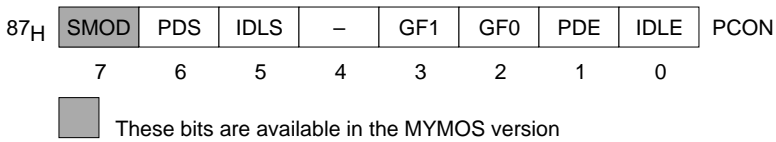
Chapters 7.1.1.1, 7.1.1.2, 7.1.1.3 are dealing with the port structures in detail.

2.1.4 The A/D Converter Input Ports

The analog input ports (AN0 to AN7) of the SAB 80515/80535 can only be used as analog inputs for the A/D converter.

The analog input ports (P6.0 to P6.7) of the SAB 80C515/80C535 can be used either as input channels for the A/D converter or as digital inputs (see chapter 7.4)

Figure 2-2
Special Function Register PCON (Address 87H)



Symbol	Position	Function
SMOD	PCON.7	When set, the baud rate of the serial channel in mode 1, 2, 3 is doubled.
PDS	PCON.6	Power-down start bit. The instruction that sets the PDS flag bit is the last instruction before entering the power-down mode.
IDLS	PCON.5	Idle start bit. The instruction that sets the IDLS flag bit is the last instruction before entering the idle mode.
–	PCON.4	Reserved
GF1	PCON.3	General purpose flag
GF0	PCON.2	General purpose flag
PDE	PCON.1	Power-down enable bit. When set, starting of the power-down mode is enabled.
IDLE	PCON.0	Idle mode enable bit. When set, starting of the idle mode is enabled.

2.1.5 A/D Converter Timings

See the corresponding data sheets for the specification of t_L (load time), t_S (sample time), t_C (conversion time).

2.1.6 The Oscillator and Clock Circuits

There is no difference between the MYMOS and ACMOS versions if they are driven from a crystal or a ceramic resonator.

Please note that there is a difference between driving MYMOS and ACMOS components from external source. How to drive each device is described in chapter 7.8.2 and in each data sheet.

2.1.7 The V_{BB} Pin

The SAB 80515/80535 has an extra V_{BB} pin connected to the device's substrate. It must be connected to V_{SS} through a capacitor for proper operation of the A/D converter.

The SAB 80C515/80C535 has no V_{BB} pin. In ACMOS technology the substrate is directly connected to V_{CC} ; therefore, the corresponding pin is used as an additional V_{CC} pin.

3 Central Processing Unit

3.1 General Description

The CPU (Central Processing Unit) of the SAB 80(C)515 consists of the instruction decoder, the arithmetic section and the program control section. Each program instruction is decoded by the instruction decoder. This unit generates the internal signals controlling the functions of the individual units within the CPU. They have an effect on the source and destination of data transfers, and control the ALU processing.

The arithmetic section of the processor performs extensive data manipulation and is comprised of the Arithmetic/Logic Unit (ALU), an A register, B register and PSW register. The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under the control of the instruction decoder. The ALU performs the arithmetic operations add, subtract, multiply, divide, increment, decrement, BCD-decimal-add-adjust and compare, and the logic operations AND, OR, Exclusive OR, complement and rotate (right, left or swap nibble (left four)). Also included is a Boolean processor performing the bit operations of set, clear, complement, jump-if-not-set, jump-if-set-and-clear and move to/from carry. Between any addressable bit (or its complement) and the carry flag, it can perform the bit operations of logical AND or logical OR with the result returned to the carry flag. The A, B and PSW registers are described in section 4.4.

The program control section controls the sequence in which the instructions stored in program memory are executed. The 16-bit program counter (PC) holds the address of the next instruction to be executed. The PC is manipulated by the control transfer instructions listed in the chapter "Instruction Set". The conditional branch logic enables internal and external events to the processor to cause a change in the program execution sequence.

3.2 CPU Timing

A machine cycle consists of 6 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 (state 1, phase 1) through S6P2 (state 6, phase 2). 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 3-1** show the fetch/execute timing related to the internal states and phases. Since these internal clock signals are not user-accessible, the XTAL2 oscillator signals and the ALE (address latch enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the op-code is latched into the instruction register. If it is a two-byte instruction, the second is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a fetch at S4, but the byte read (which would be the next op-code) is ignored, and the program counter is not incremented. In any case, execution is completed at the end of S6P2.

Figures 3-1 A) and B) show the timing of a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most SAB 80(C)515 instructions are executed 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 the program memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a one-byte, 2-cycle instruction that accesses external data memory. During a MOVX, the two fetches in the second cycle are skipped while the external data memory is being addressed and strobed. **Figures 3-1 C) and D)** show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

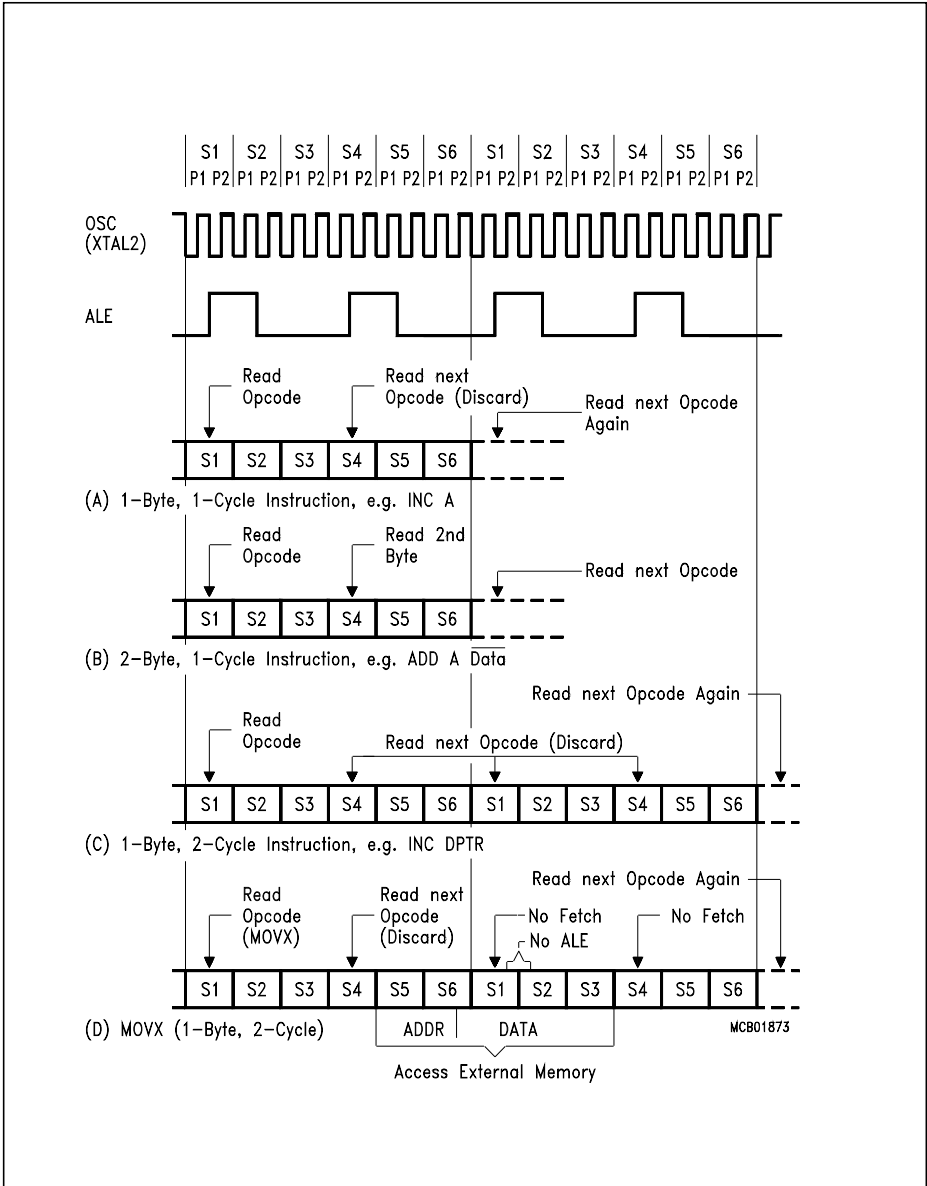


Figure 3-1
Fetch/Execute Sequence

4 Memory Organization

The SAB 80(C)515 CPU manipulates operands in the following four address spaces:

- up to 64 Kbyte of program memory
- up to 64 Kbyte of external data memory
- 256 bytes of internal data memory
- a 128-byte special function register area

4.1 Program Memory

The program memory of the SAB 80(C)515 consists of an internal and an external memory portion (see figure 4-1). 8 Kbyte of program memory may reside on-chip (SAB 80C515/80515 only), while the SAB 80C535/80535 has no internal ROM. The program memory can be externally expanded up to 64 Kbyte. If the \overline{EA} pin is held high, the SAB 80(C)515 executes out of the internal program memory unless the address exceeds $1FFF_H$. Locations 2000_H through $0FFFF_H$ are then fetched from the external memory. If the \overline{EA} pin is held low, the SAB 80(C)515 fetches all instructions from the external program memory. Since the SAB 80C535/80535 has no internal program memory, pin \overline{EA} must be tied low when using this device. In either case, the 16-bit program counter is the addressing mechanism.

Locations 03_H through 93_H in the program memory are used by interrupt service routines.

4.2 Data Memory

The data memory address space consists of an internal and an external memory portion.

Internal Data Memory

The internal data memory address space is divided into three physically separate and distinct blocks: the lower 128 bytes of RAM, the upper 128-byte RAM area, and the 128-byte special function register (SFR) area (see figure 4-2). Since the latter SFR area and the upper RAM area share the same address locations, they must be accessed through different addressing modes. The map in figure 4-2 and the following table show the addressing modes used for the different RAM/SFR spaces.

Address Space	Locations	Addressing Mode
Lower 128 bytes of RAM	00 _H to 7F _H	direct/indirect
Upper 128 bytes of RAM	80 _H to 0FF _H	indirect
Special function registers	80 _H to 0FF _H	direct

For details about the addressing modes see chapter 9.1.

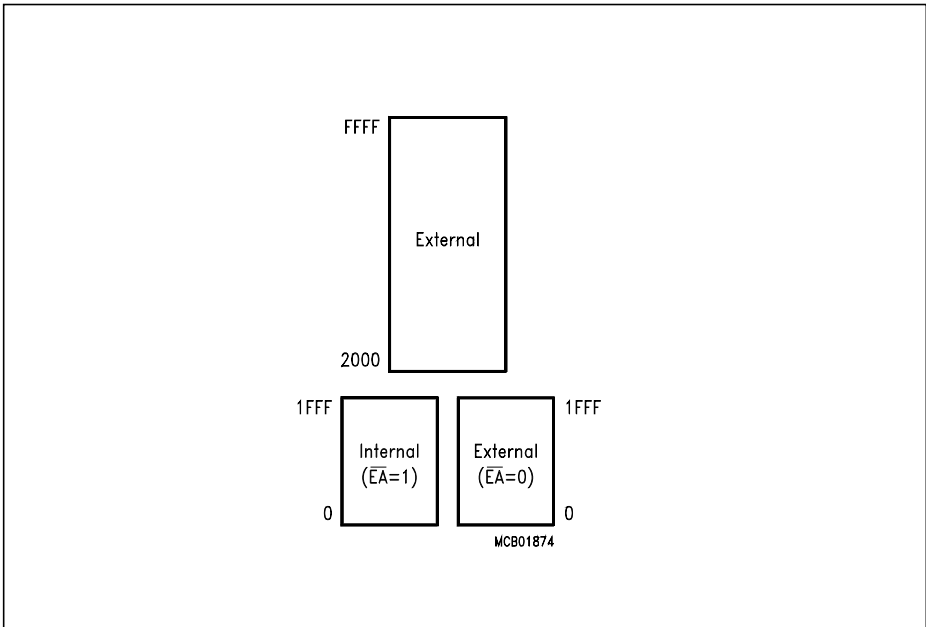


Figure 4-1
Program Memory Address Space

The lower 128 bytes of the internal RAM are again grouped in three address spaces (see figure 4-3):

- 1) A general purpose register area occupies locations 0 through 1F_H (see also section 4.3).
- 2) The next 16 bytes, location 20_H through 2F_H, contain 128 directly addressable bits. Programming information: These bits can be referred to in two ways, both of which are acceptable for the ASM51. One way is to refer to their bit addresses, i.e. 0 to 7F_H. The other way is by referencing to bytes 20_H to 2F_H. Thus bits 0 to 7 can also be referred to as bits 20.0 to 20.7, and bits 08_H and 0F_H are the same as 21.0 to 21.7 and so on. Each of the 16 bytes in this segment may also be addressed as a byte.)
- 3) Locations 30_H to 7F_H can be used as a scratch pad area.

Using the Stack Pointer (SP) - a special function register described in section 4.4 - the stack can be located anywhere in the whole internal data memory address space. The stack depth is limited only by the internal RAM available (256 byte maximum). However, the user has to take care that the stack is not overwritten by other data, and vice versa.

External Data Memory

Figure 4-2 and 4-3 contain memory maps which illustrate the internal/external data memory. To address data memory external to the chip, the "MOVX" instructions in combination with the 16-bit datapointer or an 8-bit general purpose register are used. Refer to chapter 9 (Instruction Set) or 5 (External Bus Interface) for detailed descriptions of these operations. A maximum of 64 Kbytes of external data memory can be accessed by instructions using 16-bit address.

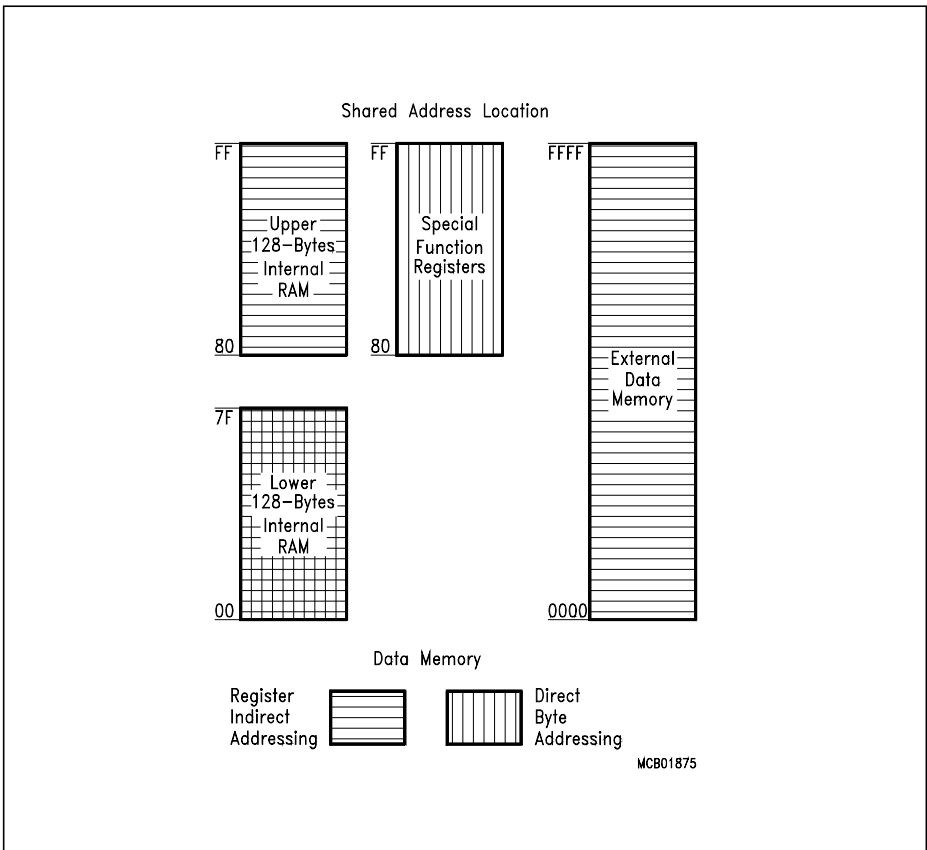


Figure 4-2
Data Memory / SFR Address Space

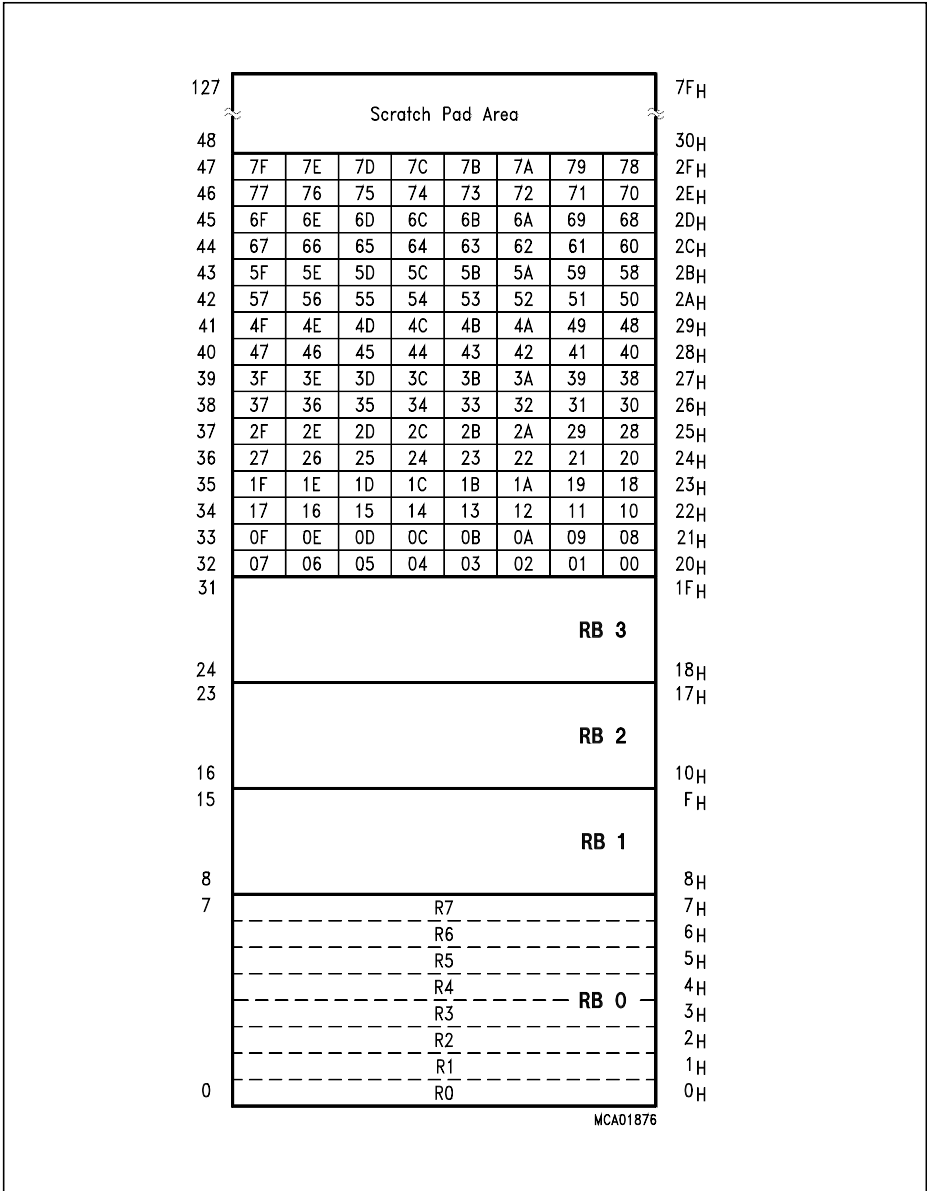


Figure 4-3 Mapping of the Lower Portion of the Internal Data Memory

4.3 General Purpose Register

The lower 32 locations of the internal RAM are assigned to four banks with eight general purpose register (GPRs) each. Only one of these banks may be enabled at a time. Two bits in the program status word, PSW.3 and PSW.4, select the active register bank (see description of the PSW). This allows fast context switching, which is useful when entering subroutines or interrupt service routines. ASM51 and the device SAB 80(C)515 default to register bank 0.

The 8 general purpose registers of the selected register bank may be accessed by register addressing. With register addressing the instruction op code indicates which register is to be used. For indirect accessing R0 and R1 are used as pointer or index register to address internal or external memory (e.g. MOV @R0).

Reset initializes the stack pointer to location 07H and increments it once to start from location 08H which is also the first register (R0) of register bank 1. Thus, if one is going to use more than one register bank, the SP should be initiated to a different location of the RAM which is not used for data storage.

4.4 Special Function Registers

The Special Function Register (SFR) area has two important functions. Firstly, all CPU register except the program counter and the four register banks reside here. The CPU registers are the arithmetic registers like A, B, PSW and pointers like SP, DPH and DPL.

Secondly, a number of registers constitute the interface between the CPU and all on-chip peripherals. That means, all control and data transfers from and to the peripherals use this register interface exclusively.

The special function register area is located in the address space above the internal RAM from addresses 80H to FFH. All 41 special function registers of the SAB 80(C)515 reside here.

Fifteen SFRs, that are located on addresses dividable by eight, are bit-addressable, thus allowing 128 bit-addressable locations within the SFR area.

Since the SFR area is memory mapped, access to the special function registers is as easy as with internal RAM, and they may be processed with most instructions. In addition, if the special functions are not used, some of them may be used as general scratch pad registers. Note, however, all SFRs can be accessed by direct addressing only.

The special function registers are listed in **table 4-1**. Bit- and byte-addressable special function registers are marked with an asterisk at the symbol name.

Table 4-1
Special Function Registers

Symbol	Name	Address
* P0	Port 0	80 _H
SP	Stack pointer	81 _H
DPL	Data pointer, low byte	82 _H
DPH	Data pointer, high byte	83 _H
PCON	Power control register	87 _H
* TCON	Timer control register	88 _H
TMOD	Timer mode register	89 _H
TL0	Timer 0, low byte	8A _H
TL1	Timer 1, low byte	8B _H
TH0	Timer 0, high byte	8C _H
TH1	Timer 1, high byte	8D _H
* P1	Port 1	90 _H
* SCON	Serial channel control register	98 _H
SBUF	Serial channel buffer register	99 _H
* P2	Port 2	0A0 _H
* IEN0	Interrupt enable register 0	0A8 _H
IP0	Interrupt priority register 0	0A9 _H
* P3	Port 3	0B0 _H
* IEN1	Interrupt enable register 1	0B8 _H
IP1	Interrupt priority register 1	0B9 _H
* IRCON	Interrupt request control register	0C0 _H
CCEN	Compare/capture enable register	0C1 _H
CCL1	Compare/capture register 1, low byte	0C2 _H
CCH1	Compare/capture register 1, high byte	0C3 _H
CCL2	Compare/capture register 2, low byte	0C4 _H
CCH2	Compare/capture register 2, high byte	0C5 _H
CCL3	Compare/capture register 3, low byte	0C6 _H
CCH3	Compare/capture register 3, high byte	0C7 _H
* T2CON	Timer 2 control register	0C8 _H
CRCL	Compare/reload/capture register, low byte	0CA _H
CRCH	Compare/reload/capture register, high byte	0CB _H
TL2	Timer 2, low byte	0CC _H
TH2	Timer 2, high byte	0CD _H
* PSW	Program status word register	0D0 _H
* ADCON	A/D converter control register	0D8 _H
ADDAT	A/D converter data register	0D9 _H
DAPR	D/A converter program register	0DA _H
P6	Port 6	0DB _H ¹⁾
* ACC	Accumulator	0E0 _H
* P4	Port 4	0E8 _H
* B	B register	0F0 _H
* P5	Port 5	0F8 _H

The SFR's marked with an asterisk (*) are bit- and byte-addressable.

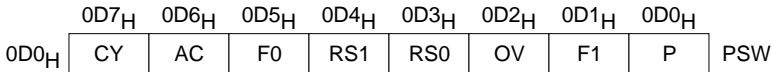
1) Additional feature of the ACMOS versions

The following paragraphs give a general overview of the special function register and refer to sections where a more detailed description can be found.

Accumulator, SFR Address 0E0_H

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

Figure 4-4
Program Status Word Register (PSW), SFR Address 0D0_H



The PSW register contains program status information.

Bit	Function
CY	Carry flag
AC	Auxiliary carry flag (for BCD operations)
F0	General purpose user flag 0
RS1 RS0	Register bank select control bits
0 0	Bank 0 selected, data address 00 _H - 07 _H
0 1	Bank 1 selected, data address 08 _H - 0F _H
1 0	Bank 2 selected, data address 10 _H - 17 _H
1 1	Bank 3 selected, data address 18 _H - 1F _H
OV	Overflow flag
F1	General purpose user flag
P	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.

B Register, SPF Address 0F0_H

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

Stack Pointer, SFR Address 081_H

The Stack Pointer (SP) register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions and decremented after data is popped during a POP and RET (RETI) execution, i.e. it always points to the last valid stack byte. While the stack may reside anywhere in on-chip RAM, the stack pointer is initialized to 07_H after a reset. This causes the stack to begin at location 08_H above register bank zero. The SP can be read or written under software control.

Datapointer, SFR Address 082H and 083H

The 16-bit Datapointer (DPTR) register is a concatenation of registers DPH (data pointer's high order byte) and DPL (data pointer's low order byte). The data pointer is used in register-indirect addressing to move program memory constants and external data memory variables, as well as to branch within the 64 Kbyte program memory address space.

Ports 0 to 5

P0 to P5 are the SFR latches to port 0 to 5, respectively. The port SFRs 0 to 5 are bit-addressable. Ports 0 to 5 are 8-bit I/O ports (that is in total 48 I/O lines) which may be used as general purpose ports and which provide alternate output functions dedicated to the on-chip peripherals of the SAB 80(C)515.

Port 6 (AN0 to AN7)

In the MYMOS versions, the analog input lines AN0 to AN7 can only be used as inputs for the A/D converter.

In the ACMCS versions these lines may also be used as digital inputs. In this case they are addressed as an additional input port (port 6) via special function register P6 (0DBH). Since port 6 has no internal latch, the contents of SFR P6 only depends on the levels applied to the input lines.

For details about this port please refer to section 7.1 (Parallel I/O).

Peripheral Control, Data and Status Registers

Most of the special function registers are used as control, status, and data registers to handle the on-chip peripherals.

In the special function register table the register names are organized in groups and each of these groups refer to one peripheral unit. More details on how to program these registers are given in the descriptions of the following peripheral units:

Unit	Symbol	Section
Ports	–	7.1
Serial Channel	–	7.2
Timer 0/1	–	7.3
A/D-Converter	ADC	7.4
Timer 2 with Comp/Capt/Reload	CCU	7.5
Power Saving Modes	–	7.6
Watchdog Timer	WDT	7.7
Interrupt System	–	8

5 External Bus Interface

The SAB 80(C)515 allows for external memory expansion. To accomplish this, the external bus interface common to most 8051-based controllers is employed.

5.1 Accessing External Memory

It is possible to distinguish between accesses to external program memory and external data memory or other peripheral components respectively. This distinction is made by hardware: accesses to external program memory use the signal $\overline{\text{PSEN}}$ (program store enable) as a read strobe. Accesses to external data memory use $\overline{\text{RD}}$ and $\overline{\text{WR}}$ to strobe the memory (alternate functions of P3.7 and P3.6, see section 7.1.). Port 0 and port 2 (with exceptions) are used to provide data and address signals. In this section only the port 0 and port 2 functions relevant to external memory accesses are described (for further details see chapter 7.1).

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).

Role of P0 and P2 as Data/Address Bus

When used for accessing external memory, port 0 provides the data byte time-multiplexed with the low byte of the address. In this state, port 0 is disconnected from its own port latch, and the address/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 pullup resistors.

During any access to external memory, the CPU writes 0FF_{H} to the port 0 latch (the special function register), thus obliterating whatever information the port 0 SFR may have been holding.

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. During this time, the port 2 lines are disconnected from the port 2 latch (the special function register).

Thus the port 2 latch does not have to contain 1 s, and the contents of the port 2 SFR are not modified.

If an 8-bit address is 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. It should be noted that, if a port 2 pin outputs an address bit that is a 1, strong pullups will be used for the entire read/write cycle and not only for two oscillator periods.

Timing

The timing of the external bus interface, in particular the relationship between the control signals ALE, $\overline{\text{PSEN}}$, $\overline{\text{RD}}$ and information on port 0 and port 2, is illustrated in **figure 5-1 a)** and **b)**.

Data memory: in a write cycle, the data byte to be written appears on port 0 just before $\overline{\text{WR}}$ is activated, and remains there until after $\overline{\text{WR}}$ is deactivated. In a read cycle, the incoming byte is accepted at port 0 before the read strobe is deactivated.

Program memory: signal $\overline{\text{PSEN}}$ functions as a read strobe. For further information see section 5.2.

External Program Memory Access

The external program memory is accessed under two conditions:

- whenever signal $\overline{\text{EA}}$ is active; or
- whenever the program counter (PC) contains a number that is larger than 01FFF_{H} .

This requires the ROM-less versions SAB 80C535/80535 to have $\overline{\text{EA}}$ wired low to allow the lower 8 K 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. The contents of the port 2 SFR however is not affected. During external program memory fetches port 2 lines output the high byte of the PC, and during accesses to external data memory they output either DPH or the port 2 SFR (depending on whether the external data memory access is a MOVX @DPTR or a MOVX @Ri).

Since the SAB 80C535/80535 has no internal program memory, accesses to program memory are always external, and port 2 is at all times dedicated to output the high-order address byte. This means that port 0 and port 2 of the SAB 80C535/80535 can never be used as general-purpose I/O. This also applies to the SAB 80C515/80515 when it is operated with only an external program memory.

5.2 $\overline{\text{PSEN}}$, Program Store Enable

The read strobe for external fetches is $\overline{\text{PSEN}}$. $\overline{\text{PSEN}}$ is not activated for internal fetches. When the CPU is accessing external program memory, $\overline{\text{PSEN}}$ is activated twice every cycle (except during a MOVX instruction) no matter whether or not the byte fetched is actually needed for the current instruction. When $\overline{\text{PSEN}}$ is activated its timing is not the same as for $\overline{\text{RD}}$. A complete $\overline{\text{RD}}$ cycle, including activation and deactivation of ALE and $\overline{\text{RD}}$, takes 12 oscillator periods. A complete $\overline{\text{PSEN}}$ cycle, including activation and deactivation of ALE and $\overline{\text{PSEN}}$ takes 6 oscillator periods. The execution sequence for these two types of read cycles is shown in **figure 5-1 a)** and **b)**.

5.3 ALE, Address Latch Enable

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 into an external latch during fetches from external memory. The address byte is valid at the negative transition of ALE. For that purpose, ALE is activated twice every machine cycle. This activation takes place even if the cycle involves no external fetch. The only time no ALE pulse comes out is during an access to external data memory when $\overline{\text{RD}}/\overline{\text{WR}}$ signals are active. The first ALE of the second cycle of a MOVX instruction is missing (see **figure 5-1b**). Consequently, in any system that does not use data memory, ALE is activated at a constant rate of 1/6 of the oscillator frequency and can be used for external clocking or timing purposes.

5.4 Overlapping External Data and Program Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is used for storing data. In the SAB 80(C)515, the external program and data memory spaces can be combined by AND-ing $\overline{\text{PSEN}}$ and $\overline{\text{RD}}$. 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 $\overline{\text{PSEN}}$ cycle is faster than the $\overline{\text{RD}}$ cycle, the external memory needs to be fast enough to adapt to the $\overline{\text{PSEN}}$ cycle.

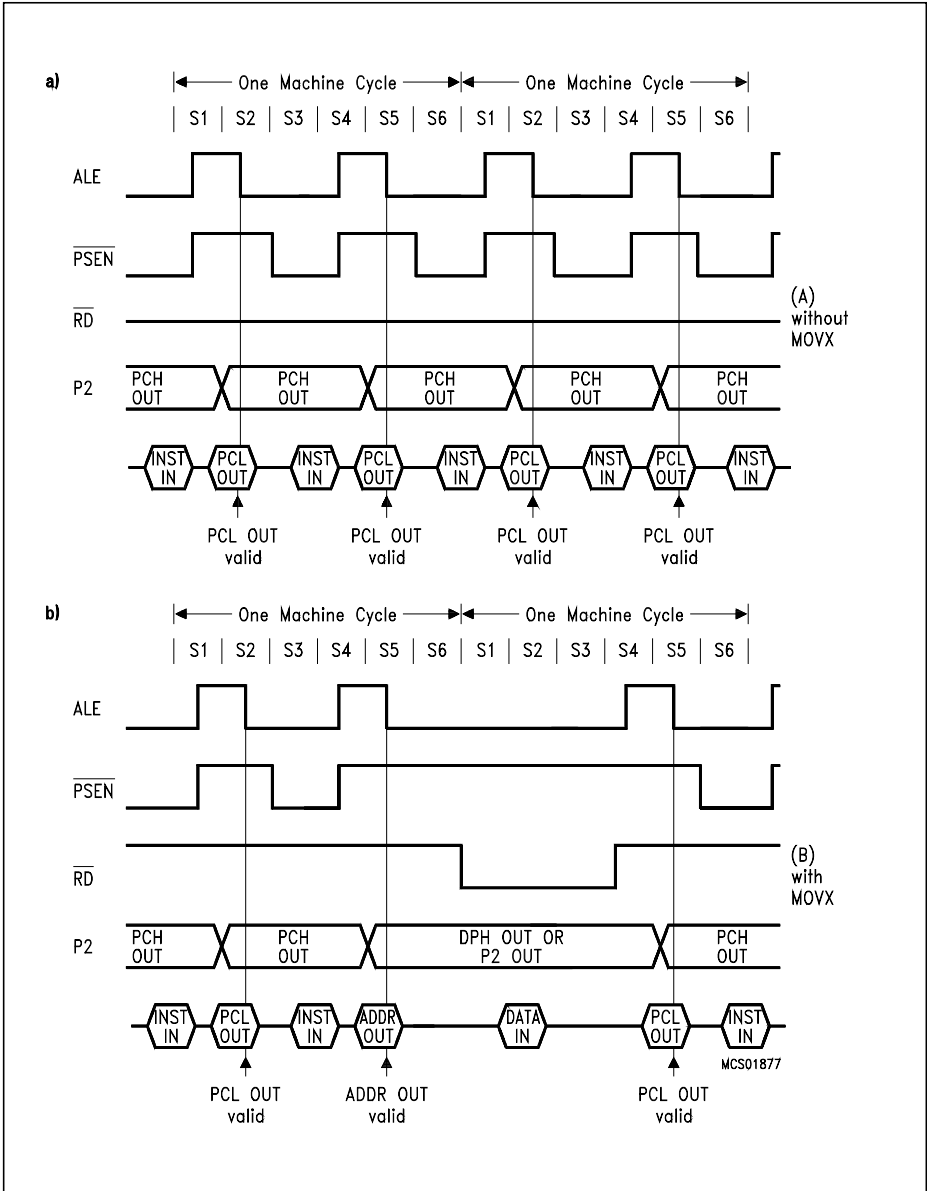


Figure 5-1 a) and b)
 External Program Memory Execution

6 System Reset

6.1 Hardware Reset and Power-Up Reset

6.1.1 Reset Function and Circuitries

The hardware reset function incorporated in the SAB 80(C)515 allows for an easy automatic start-up at a minimum of additional hardware and forces the controller to a predefined default state. The hardware reset function can also be used during normal operation in order to restart the device. This is particularly done when the power-down mode (see section 7.6) is to be terminated.

In addition to the hardware reset, which is applied externally to the SAB 80(C)515, there is also the possibility of an internal hardware reset. This internal reset will be initiated by the watchdog timer (section 7.7).

The reset input is an active low input at pin 10 ($\overline{\text{RESET}}$). An internal Schmitt trigger is used at the input for noise rejection. Since the reset is synchronized internally, the $\overline{\text{RESET}}$ pin must be held low for at least two machine cycles (24 oscillator periods) while the oscillator is running. With the oscillator running the internal reset is executed during the second machine cycle in which $\overline{\text{RESET}}$ is low and is repeated every cycle until $\overline{\text{RESET}}$ goes high again.

During reset, pins ALE and $\overline{\text{PSEN}}$ are configured as inputs and should not be stimulated externally. (An external stimulation at these lines during reset activates several test modes which are reserved for test purposes. This in turn may cause unpredictable output operations at several port pins).

A pullup resistor is internally connected to V_{CC} to allow a power-up reset with an external capacitor only. An automatic reset can be obtained when V_{CC} is applied by connecting the reset pin to V_{SS} via a capacitor as shown in **figure 6-1 a)** and **c)**. After V_{CC} has been turned on the capacitor must hold the voltage level at the reset pin for a specified time below the upper threshold of the Schmitt trigger to effect a complete reset.

The time required is the oscillator start-up time plus 2 machine cycles, which, under normal conditions, must be at least 10 - 20 ms for a crystal oscillator. This requirement is usually met using a capacitor of 4.7 to 10 microfarad. The same considerations apply if the reset signal is generated externally (**figure 6-1 b)**). In each case it must be assured that the oscillator has started up properly and that at least two machine cycles have passed before the reset signal goes inactive.

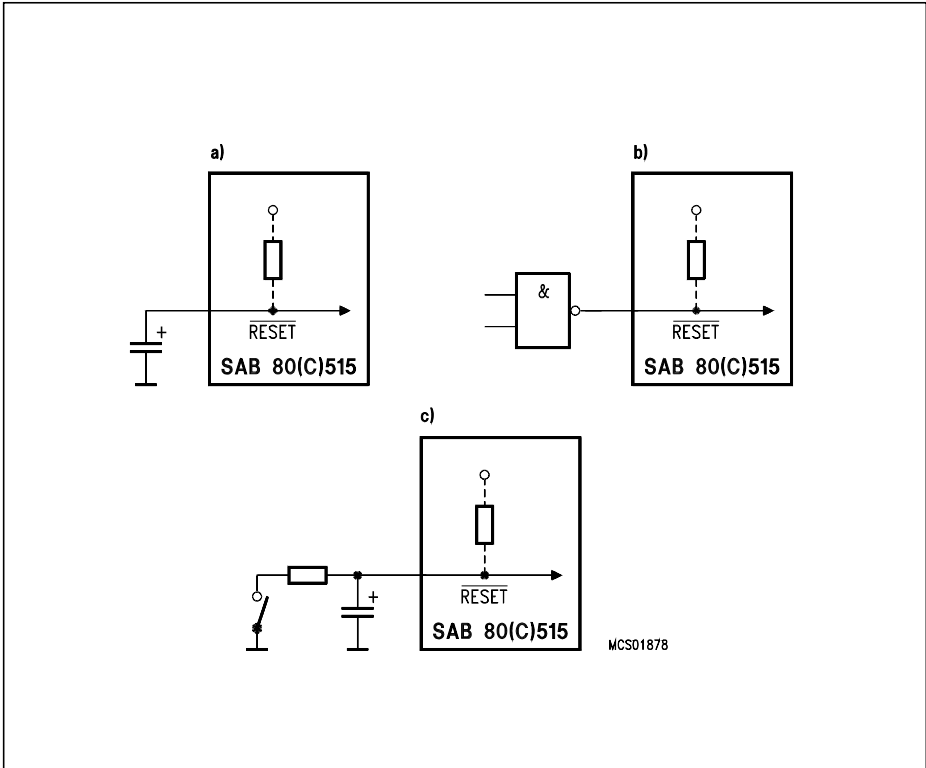


Figure 6-1 a) - c)
Reset Circuitries

A correct reset leaves the processor in a defined state. The program execution starts at location 0000_H. The default values of the special function registers (SFR) during and after reset are listed in **table 6-1**. After reset is internally accomplished the contents of the port latches of port 0 to 5 is 0FF_H. This leaves port 0 floating, since it is an open drain port when not used as data/address bus. All other I/O port lines (ports 1 through 5) output a one (1).

In the MYMOS versions, the analog input lines AN0 to AN7 can only be used as inputs.

In the ACMOS versions these lines may also be used as digital inputs. In this case they are addressed as an additional input port (port 6) via special function register P6 (0DB_H). Since port 6 has no internal latch, the contents of SFR P6 only depends on the levels applied to the input lines.

For details about this port please refer to section 7.1 (Parallel I/O).

The contents of the internal RAM of the SAB 80(C)515 is not affected by a reset. After power-up the contents is undefined, while it remains unchanged during a reset if the power supply is not turned off.

Table 6-1
Register Contents after Reset

Register	Contents	Register	Contents
P0 - P5	0FF _H	SP	07 _H
DPTR	0000 _H	PCON	000X 0000B
TCON	00 _H	TMOD	00 _H
TL0, TH0	00 _H	TL1, TH1	00 _H
TL2, TH2	00 _H	SCON	00 _H
IEN0, IEN1	00 _H	SBUF	undefined
IRCON	00 _H	IP0	X000 0000B
		IP1	XX00 0000B
CCL1, CCH1	00 _H	CCEN	00 _H
CCL3, CCH3	00 _H	CCL2, CCH2	00 _H
T2CON	00 _H	CRCL, CRCH	00 _H
ADCON	00X0 0000B	PSW	00 _H
DAPR	00 _H	ADDAT	00 _H
B	00 _H	ACC	00 _H
PC	0000 _H	Watchdog	0000 _H

6.1.2 Hardware Reset Timing

This section describes the timing of the hardware reset signal.

The input pin $\overline{\text{RESET}}$ is sampled once during each machine cycle. This happens in state 5 phase 2. Thus, the external reset signal is synchronized to the internal CPU timing. When the reset is found active (low level at pin 10) the internal reset procedure is started. It needs two complete machine cycles to put the complete device to its correct reset state. i.e. all special function registers contain their default values, the port latches contain 1's etc. Note that this reset procedure is not performed if there is no clock available at the device. The $\overline{\text{RESET}}$ signal must be active for at least two machine cycles; after this time the SAB 80(C)515 remains in its reset state as long as the signal is active. When the signal goes inactive this transition is recognized in the following state 5 phase 2 of the machine cycle. Then the processor starts its address output (when configured for external ROM) in the following state 5 phase 1. One phase later (state 5 phase 2) the first falling edge at pin ALE occurs. **Figure 6-2** shows this timing for a configuration with $\overline{\text{EA}} = 0$ (external program memory). Thus, between the release of the $\overline{\text{RESET}}$ signal and the first falling edge at ALE there is a time period of at least one machine cycle but less than two machine cycles.

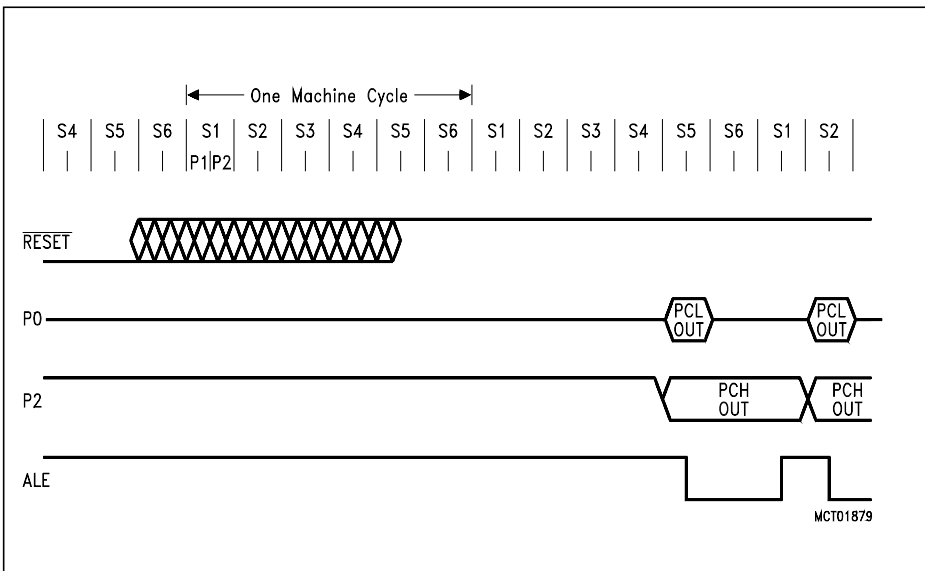


Figure 6-2
CPU Timing after RESET

7 On-Chip Peripheral Components

This chapter gives detailed information about all on-chip peripherals of the SAB 80(C)515 except for the integrated interrupt controller, which is described separately in chapter 8. Sections 7.1 and 7.2 are associated with the general parallel and serial I/O facilities while the remaining sections describe the miscellaneous functions such as the timers, serial interface, A/D converter, power saving modes, watchdog timer, oscillator and clock circuitries, and system clock output.

7.1 Parallel I/O

7.1.1 Port Structures

Digital I/O

The SAB 80(C)515 allows for digital I/O on 48 lines grouped into 6 bidirectional 8-bit ports. Each port bit consists of a latch, an output driver and an input buffer. Read and write accesses to the I/O ports P0 through P5 are performed via their corresponding special function registers P0 to P5.

The output drivers of port 0 and 2 and the input buffers of port 0 are also used for accessing 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 memory address when the address is 16 bits wide. Otherwise, the port 2 pins continue emitting the P2 SFR contents (see also chapter 7.1.2 and chapter 5 for more details about the external bus interface).

Digital/Analog Input Ports

The analog input lines AN0 to AN7 of the MYMOS versions can only be used as analog inputs.

In the ACMOS versions these lines may also be used as digital inputs. In this case they are addressed as an additional input port (port 6) via special function register P6 (0DB_H). Since port 6 has no internal latch, the contents of SFR P6 only depends on the levels applied to the input lines.

When used as analog input the required analog channel is selected by a three-bit field in SFR ADCON, as described in section 7.4. Of course, it makes no sense to output a value to these input-only ports by writing to the SFR P6 or P8; this will have no effect.

If a digital value is to be read, the voltage levels are to be held within the input voltage specifications (V_{IL}/V_{IH}). Since P6 is not a bit-addressable register, all input lines of P6 are read at the same time by byte instructions.

Nevertheless, it is possible to use port 6 simultaneously for analog and digital input. However, care must be taken that all bits of P6 are masked which have an undetermined value caused by their analog function .

In order to guarantee a high-quality A/D conversion, digital input lines of port 6 should not toggle while a neighbouring port pin is executing an A/D conversion. This could produce crosstalk to the analog signal.

7.1.1.1 Digital I/O Port Circuitry (MYMOS/ACMOS)

Figure 7-1 shows a functional diagram of a typical bit latch and I/O buffer, which is the core of each of the 6 I/O-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 from a port (i.e. from the corresponding port SFR P0 to P5) activate the "read-latch" signal, while others activate the "read-pin" signal (see section 7.1.4.3).

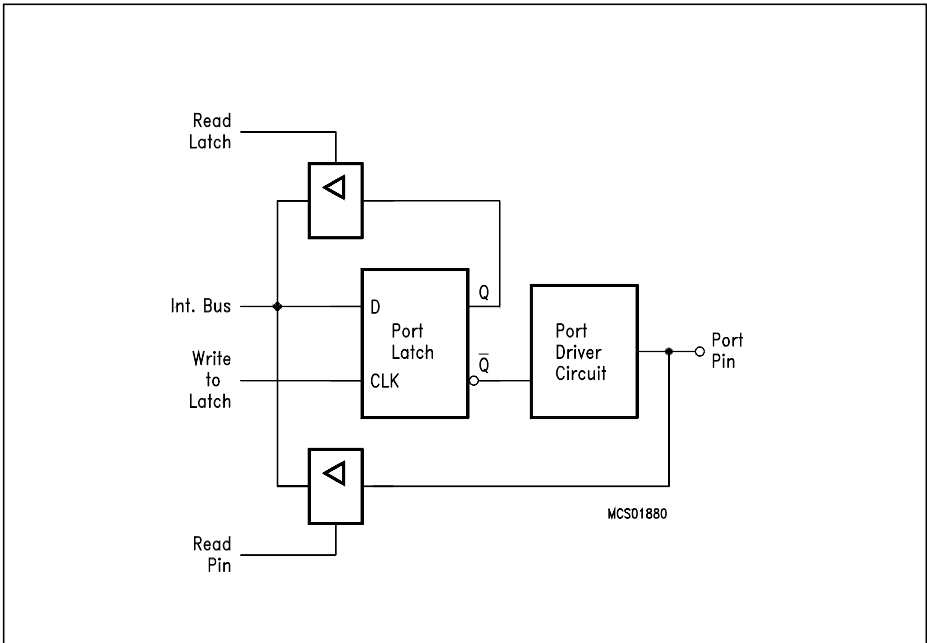


Figure 7-1
Basic Structure of a Port Circuitry

Port 1 through 5 output drivers have internal pullup FET's (see figure 7-2). Each I/O line can be used independently as an input or output. To be used as an input, the port bit must contain a one (1) (that means for figure 7-2: $\bar{Q} = 0$), which turns off the output driver FET n1. Then, for ports 1 through 5, the pin is pulled high by the internal pullups, but can be pulled low by an external source. When externally pulled low the port pins source current (I_{IL} or I_{TL}). For this reason these ports are sometimes called "quasi-bidirectional".

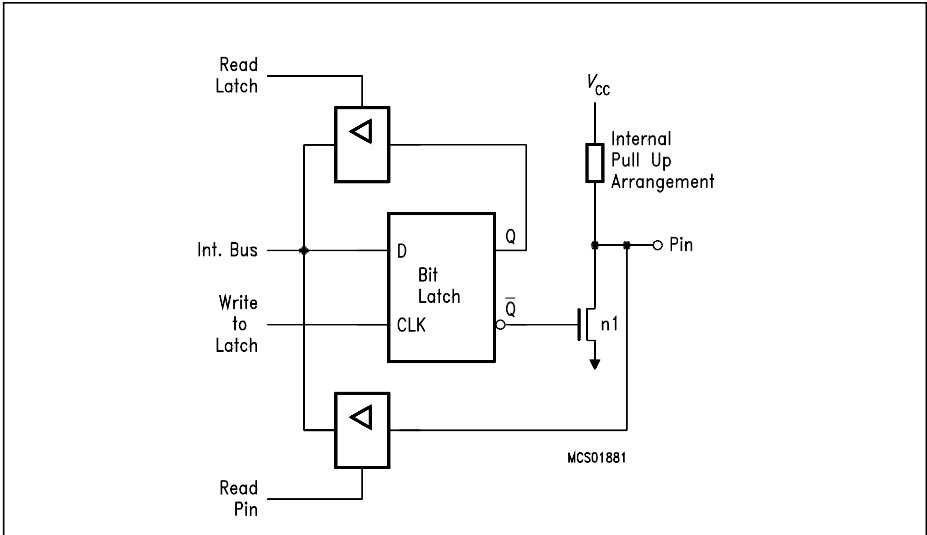


Figure 7-2
Basic Output Driver Circuit of Ports 1 through 5

In fact, the pullups mentioned before and included in figure 7-2 are pullup arrangements as shown in figure 7-3. These pullup arrangements are realized differently in the MYMOS and ACMOS versions. In the next two sections both versions are discussed separately.

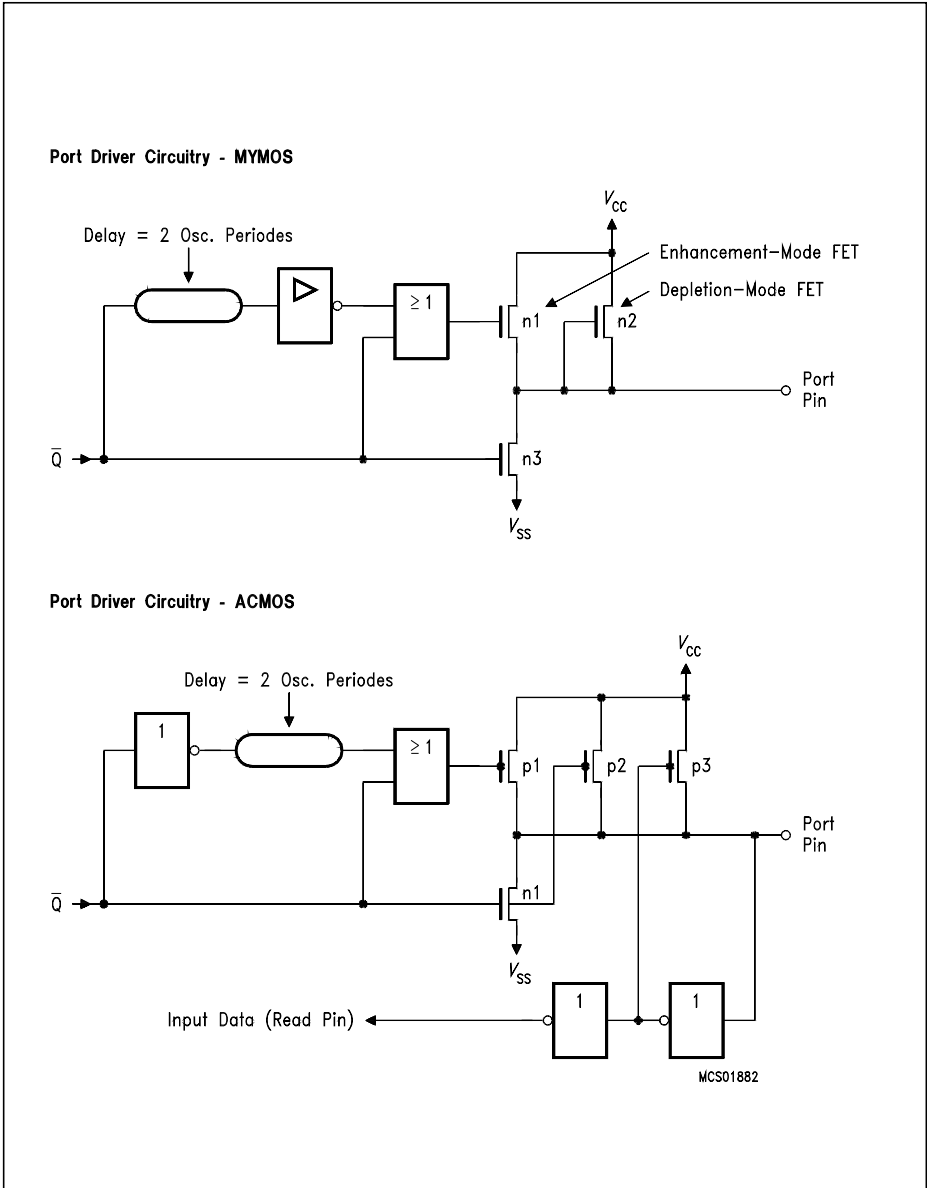


Figure 7-3
Output Driver Circuits of Ports 1 through 5

7.1.1.2 MYMOS Port Driver Circuitry

The output driver circuitry of the MYMOS version (**figure 7-3**) consists of two pullup FETs (pullup arrangements) and one pulldown FET:

- The **transistor n1** is a very strong pullup transistor which is only activated for two oscillator periods, if a 0-to-1 transition is executed by this port bit. Transistor n1 is capable of driving high currents.
- The **transistor n2** is a weak pullup transistor, which is always switched on. When the pin is pulled down (e.g. when the port is used as input), it sources a low current. This value can be found as the parameter I_{IL} in the DC characteristics.
- The **transistor n3** is a very strong pull-down transistor which is switched on when a "0" is programmed to the corresponding port latch. Transistor n3 is capable of sinking high currents (I_{OL} in the DC characteristics).
A short circuit to V_{CC} must be avoided if the transistor is turned on because the high current might destroy the FET.

7.1.1.3 ACMOS Port Driver Circuitry

The output driver circuitry of the ACMOS version (**figure 7-3**) is realized by three pullup FETs (pullup arrangement) and one pulldown FET:

- The **pulldown FET n1** is of n-channel type. It is a very strong driver transistor which is capable of sinking high currents (I_{OL}); it is only activated if a "0" is programmed to the port pin. A short circuit to V_{CC} must be avoided if the transistor is turned on, since the high current might destroy the FET.
- The **pullup FET p1** is of p-channel type. It is activated for two oscillator periods (S1P1 and S1P2) if a 0-to-1 transition is programmed to the port pin, i.e. a "1" is programmed to the port latch which contained a "0". The extra pullup can drive a similar current as the pulldown FET n1. This provides a fast transition of the logic levels at the pin.
- The **pullup FET p2** is of p-channel type. It is always activated when a "1" is in the port latch, thus providing the logic high output level. This pullup FET sources a much lower current than p1; therefore the pin may also be tied to ground, e.g. when used as input with logic low input level.
- The **pullup FET p3** is of p-channel type. It is only activated if the voltage at the port pin is higher than approximately 1.0 to 1.5 V. This provides an additional pullup current if a logic high level is to be output at the pin (and the voltage is not forced lower than approximately 1.0 to 1.5 V). However, this transistor is turned off if the pin is driven to a logic low level, e.g. when used as input. In this configuration only the weak pullup FET p2 is active, which sources the current I_{IL} . If, in addition, the pullup FET p3 is activated, a higher current can be sourced (I_{TL}). Thus, an additional power consumption can be avoided if port pins are used as inputs with a low level applied. However, the driving capability is stronger if a logic high level is output.

The described activating and deactivating of the four different transistors translates into four states the pins can be:

- input low state (IL), p2 active only
- input high state (IH) = steady output high state (SOH) p2 and p3 active
- forced output high state (FOH), p1, p2 and p3 active
- output low state (OL), n1 active

If a pin is used as input and a low level is applied, it will be in IL state, if a high level is applied, it will switch to IH state.

If the latch is loaded with "0", the pin will be in OL state.

If the latch holds a "0" and is loaded with "1", the pin will enter FOH state for two cycles and then switch to SOH state. If the latch holds a "1" and is reloaded with a "1" no state change will occur.

At the beginning of power-on reset the pins will be in IL state (latch is set to "1", voltage level on pin is below of the trip point of p3). Depending on the voltage level and load applied to the pin, it will remain in this state or will switch to IH (=SOH) state.

If it is used as output, the weak pull-up p2 will pull the voltage level at the pin above p3's trip point after some time and p3 will turn on and provide a strong "1". Note, however, that if the load exceeds the drive capability of p2 (I_{IL}), the pin might remain in the IL state and provide a weak "1" until the first 0-to-1 transition on the latch occurs. Until this the output level might stay below the trip point of the external circuitry.

The same is true if a pin is used as bidirectional line and the **external** circuitry is switched from output to input when the pin is held at "0" and the load then exceeds the p2 drive capabilities.

If the load exceeds I_{IL} the pin can be forced to "1" by writing a "0" followed by a "1" to the port pin.

Port 0, in contrast to ports 1 through 5, is considered as "true" bidirectional, because the port 0 pins float when configured as inputs. Thus, this port differs in not having internal pullups. The pullup FET in the P0 output driver (see figure 7-4 a) is used only when the port is emitting 1 s during the external memory accesses. Otherwise, the pullup is always off. Consequently, P0 lines that are used as output port lines are open drain lines. Writing a "1" to the port latch leaves both output FETs off and the pin floats. In that condition it can be used as high-impedance input. If port 0 is configured as general I/O port and has to emit logic high level (1), external pullups are required.

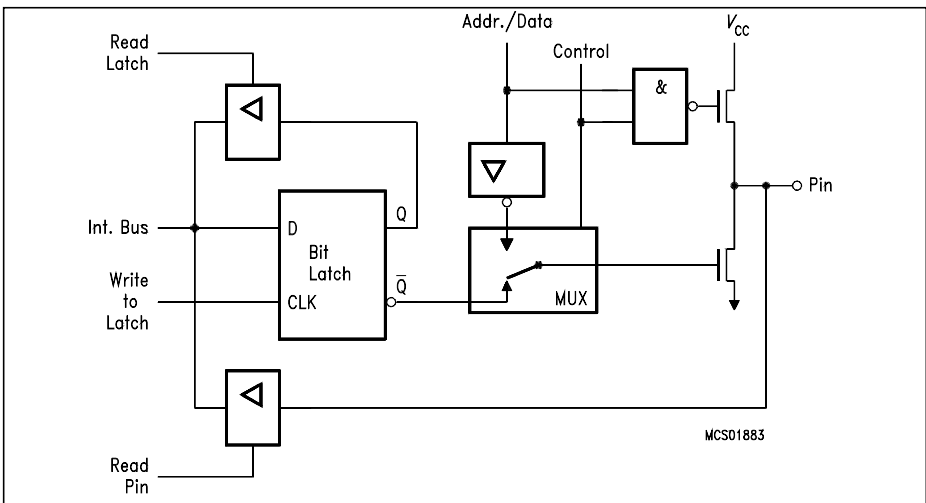


Figure 7-4 a)
Port 0 Circuitry

7.1.2 Port 0 and Port 2 Used as Address/Data Bus

As shown in **figures 7-4 a)** and **7-4 b)**, the output drivers of ports 0 and 2 can be switched to an internal address or address/data bus for use in external memory accesses. In this application they cannot be used as general purpose I/O, even if not all address lines are used externally. The switching is done by an internal control signal dependent on the input level at the \overline{EA} pin and/or the contents of the program counter. If the ports are configured as an address/data bus, the port latches are disconnected from the driver circuit. During this time, the P2 SFR remains unchanged while the P0 SFR has 1's written to it. Being an address/data bus, port 0 uses a pullup FET as shown in **figure 7-4 a)**. When a 16-bit address is used, port 2 uses the additional strong pullups p1 to emit 1's for the entire external memory cycle instead of the weak ones (p2 and p3) used during normal port activity.

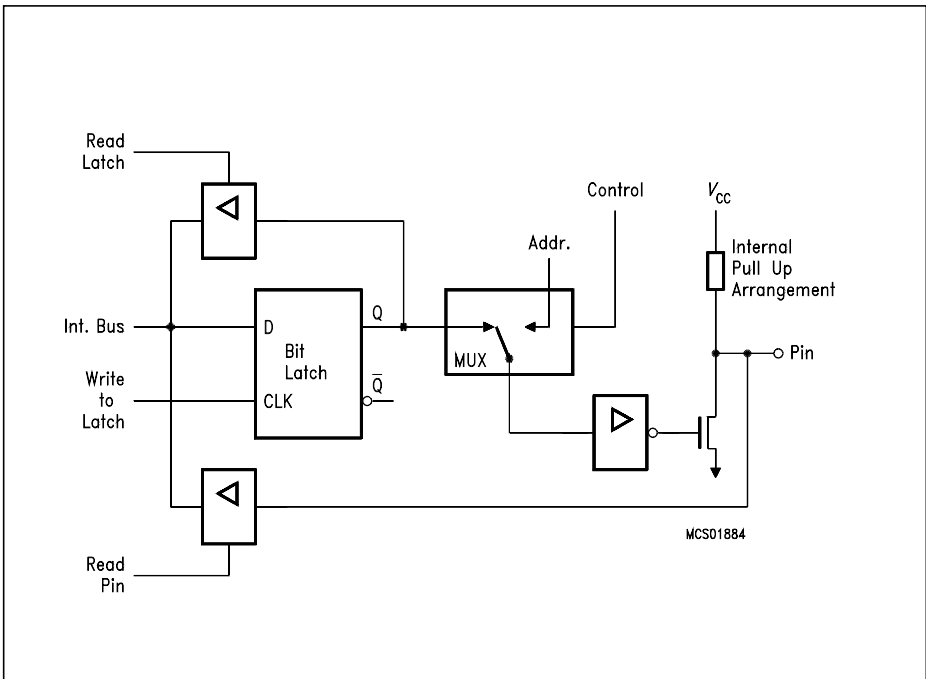


Figure 7-4 b)
Port 2 Circuitry

7.1.3 Alternate Functions

Several pins of ports 1 and 3 are multifunctional. They are port pins and also serve to implement special features as listed in **table 7-1**.

Table 7-1

Port	Pin	Alternate Function
P1.0	$\overline{\text{INT3/CC0}}$	Ext. interrupt 3 input, compare 0 output, capture 0 input
P1.1	$\overline{\text{INT4/CC1}}$	Ext. interrupt 4 input, compare 1 output, capture 1 input
P1.2	$\overline{\text{INT5/CC2}}$	Ext. interrupt 5 input, compare 2 output, capture 2 input
P1.3	$\overline{\text{INT6/CC3}}$	Ext. interrupt 6 input, compare 3 output, capture 3 input
P1.4	$\overline{\text{INT2}}$	Ext. interrupt 2 input
P1.5	T2EX	Timer 2 external reload trigger input
P1.6	CLKOUT	System clock output
P1.7	T2	Timer 2 external reload trigger input
P3.0	RXD	Serial port's receiver data input (asynchronous) or data input/output (synchronous)
P3.1	TXD	Serial port's transmitter data output (asynchronous) or clock output (synchronous)
P3.2	$\overline{\text{INT0}}$	External interrupt 0 input, timer 0 gate control
P3.3	$\overline{\text{INT1}}$	External interrupt 1 input, timer 1 gate control
P3.4	T0	Timer 0 external counter input
P3.5	T1	Timer 1 external counter input
P3.6	$\overline{\text{WR}}$	External data memory write strobe
P3.7	$\overline{\text{RD}}$	External data memory read strobe

Figure 7-5 shows a functional diagram of a port latch with alternate function. To pass the alternate function to the output pin and vice versa, however, the gate between the latch and driver circuit must be open. Thus, to use the alternate input or output functions, the corresponding bit latch in the port SFR has to contain a one (1); otherwise the pull-down FET is on and the port pin is stuck at 0. (This does not apply to ports 1.0 to 1.3 when operated in compare output mode; refer to section 7.5.2 for details). After reset all port latches contain ones (1).

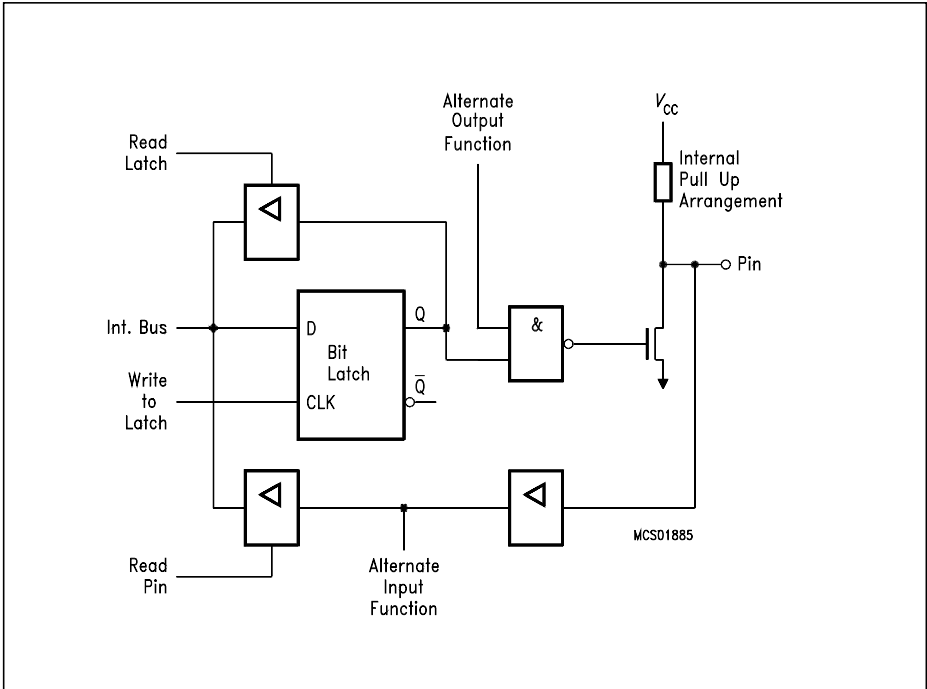


Figure 7-5
Port 1 and 3

7.1.4 Port Handling

7.1.4.1 Port Timing

When executing an instruction that changes the value of a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are only sampled by their output buffers during phase 1 of any clock period (during phase 2 the output buffer holds the value it noticed during the previous phase 1). Consequently, the new value in the port latch will not appear at the output pin until the next phase 1, which will be at S1P1 of the next machine cycle.

When an instruction reads a value from a port pin (e.g. MOV A, P1) the port pin is actually sampled in state 5 phase 1 or phase 2 depending on port and alternate functions. **Figure 7-6** illustrates this port timing. It must be noted that this mechanism of sampling once per machine cycle is also used if a port pin is to detect an "edge", e.g. when used as counter input. In this case an "edge" is detected when the sampled value differs from the value that was sampled the cycle before. Therefore, there must be met certain requirements on the pulse length of signals in order to avoid signal "edges" not being detected. The minimum time period of high and low level is one machine cycle, which guarantees that this logic level is noticed by the port at least once.

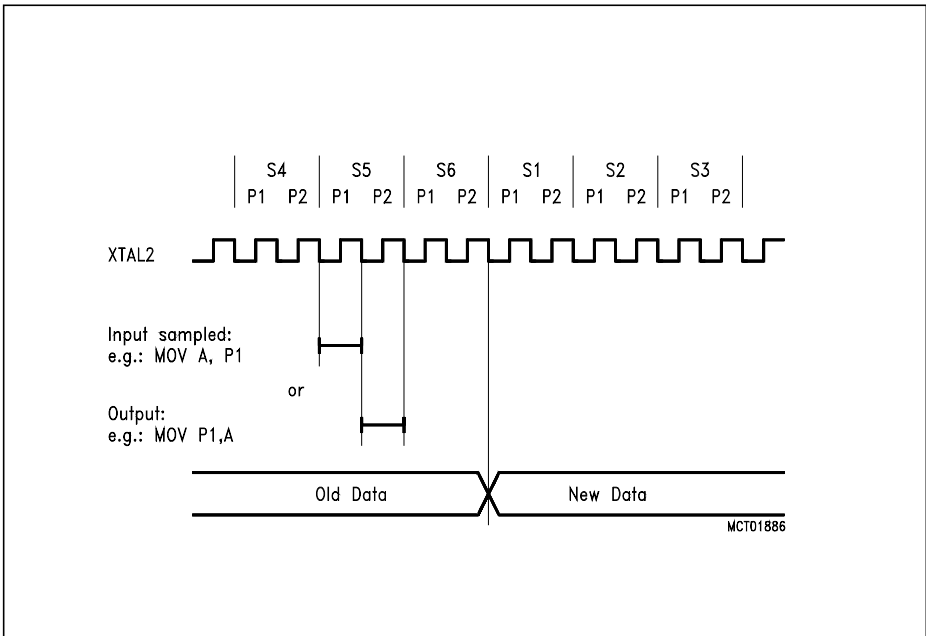


Figure 7-6
Port Timing

7.1.4.2 Port Loading and Interfacing

The output buffers of ports 1 through 5 can drive TTL inputs directly. The maximum port load which still guarantees correct logic output levels can be looked up in the DC characteristics in the Data Sheet of the SAB 80(C)515. The corresponding parameters are V_{OL} and V_{OH} .

The same applies to port 0 output buffers. They do, however, require external pullups to drive floating inputs, except when being used as the address/data bus.

When used as inputs it must be noted that the ports 1 through 5 are not floating but have internal pullup transistors. The driving devices must be capable of sinking a sufficient current if a logic low level shall be applied to the port pin (the parameters I_{TL} and I_{IL} in the DC characteristics specify these currents). Port 0 as well as the input only ports 6 of the AC MOS versions have floating inputs when used for digital input.

7.1.4.3 Read-Modify-Write Feature of Ports 0 through 5

Some port-reading instructions read the latch and others read the pin (**see figure 7-1**). The instructions reading the latch rather than the pin read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions, which are listed in **table 7-2**. If the destination is a port or a port bit, these instructions read the latch rather than the pin. Note that all other instructions which can be used to read a port, exclusively read the port pin. In any case, reading from latch or pin, resp., is performed by reading the SFR P0 to P5; for example, "MOV A, P3" reads the value from port 3 pins, while "ANL P4, #0AA_H" reads from the latch, modifies the value and writes it back to the latch.

Table 7-2
Read-Modify-Write Instructions

Instruction	Function
ANL	Logic AND; e.g. ANL P1, A
ORL	Logic OR; e.g. ORL P2, A
XRL	Logic exclusive OR; e.g. XRL P3, A
JBC	Jump if bit is set and clear bit; e.g. JBC P1.1, LABEL
CPL	Complement bit; e.g. CPL P3.0
INC	Increment byte; e.g. INC P4
DEC	Decrement byte; e.g. DEC P5
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
SETB 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. The reason is that they read the port byte, all 8 bits, modify the addressed bit, then write the complete byte back to the latch.

The reason why read-modify-write instructions are directed to the latch rather than the pin is to avoid a 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 (approx. 0.7 V, i.e. a logic low level !) and interpret it as "0". For example, when modifying a port bit by a SETB or CLR instruction, another bit in this port with the above mentioned configuration might be changed if the value read from the pin were written back to the latch. However, reading the latch rather than the pin will return the correct value of "1".

7.2 Serial Interfaces

The serial port of the SAB 80(C)515S enables communication between microcontrollers or between the microcontroller and peripheral devices.

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 has not been read by the time reception of the second byte is complete, the last received byte will be lost). The serial channel is completely compatible with the serial channel of the SAB 80(C)51.

7.2.1 Operating Modes of Serial Interface

The serial interface can operate in four modes (one synchronous mode, three asynchronous modes). The baud rate clock for this interface is derived from the oscillator frequency (mode 0, 2) or generated either by timer 1 or by a dedicated baud rate generator (mode 1, 3). A more detailed description of how to set the baud rate will follow in section 7.2.3.

Mode 0: shift register (synchronous) mode:

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

Mode 1: 8-bit UART, variable baud rate:

10 bits are transmitted (through Tx/D) or received (through Rx/D): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On reception, the stop bit goes into RB8 in special function register SCON. The baud rate is variable.

Mode 2: 9-bit UART, fixed baud rate:

11 bits are transmitted (through Tx/D) or received (through Rx/D): a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). On transmission, the 9th data bit (TB8 in SCON) can be assigned to the value of 0 or 1. For example, the parity bit (P in the PSW) could be moved into TB8 or a second stop bit by setting TB8 to 1. On reception 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 of the oscillator frequency.

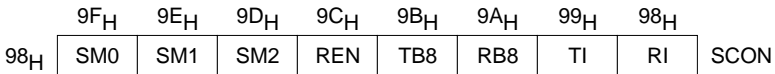
Mode 3: 9-bit UART, variable baud rate:

11 bits are transmitted (through Tx/D) or received (through Rx/D): a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). On transmission, the 9th data bit (TB8 in SCON) can be assigned to the value of 0 or 1. For example, the parity bit (P in the PSW) could be moved into TB8 or a second stop bit by setting TB8 to 1. On reception, the 9th data bit goes into RB8 in special function register SCON, while the stop bit is ignored. 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. The serial interfaces also provide interrupt requests when a transmission or a reception of a frame has completed. The corresponding interrupt request flags for serial interface are TI or RI, resp. See section 8 for more details about the interrupt structure. The interrupt request flags TI and RI can also be used for polling the serial interface if the serial interrupt is not to be used (i.e. serial interrupt not enabled).

The control and status bits of the serial channel 0 in special function register S0CON are illustrated in **figure 7-8**. **Figure 7-7** shows the special function register S0BUF which is the data register for receive and transmit. The following table summarizes the operating modes of serial interface 0.

Figure 7-7
Special Function Register SCON (Address 98H)



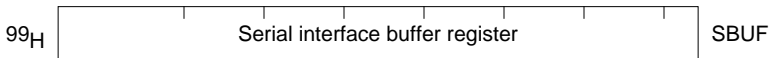
Bit	Symbol	
SM0	SM1	
0	0	Serial mode 0: Shift register mode, fixed baud rate
0	1	Serial mode 1: 8-bit UART, variable baud rate
1	0	Serial mode 2: 9-bit UART, fixed baud rate
1	1	Serial mode 3: 9-bit UART, variable baud rate
SM2		Enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3 and SM2 being set to 1, RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1 and SM2 = 1, RI will not be activated if a valid stop bit has not been received. In mode 0, SM2 should be 0.
REN		Receiver enable. Enables serial reception. Set by software to enable reception. Cleared by software to disable reception.
TB8		Transmitter bit 8. Is the 9th data bit that will be transmitted in modes 2 and 3. Set or cleared by software as desired.
RB8		Receiver bit 8. In modes 2 and 3 it is the 9th 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		Transmitter interrupt. Is the 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		Receiver interrupt. Is the receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or during the stop bit time in the other modes, in any serial reception. Must be cleared by software.

The control and status bits of the serial channel in special function register SCON are illustrated in **figure 7-8**. **Figure 7-7** shows the special function register SBUF which is the data register for receive and transmit. The following table summarizes the operating modes of the serial interface.

Table 7-3
Serial Interface, Mode Selection

SM0	SM1	Mode	Descriptions	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

Figure 7-8
Special Function Register SBUF (Address 99H)



Receive and transmit buffer of serial interface. Writing to SBUF loads the transmit register and initiates transmission. Reading out SBUF accesses a physically separate receive register.

7.2.2 Multiprocessor Communication Feature

Modes 2 and 3 of the serial interface 0 have a special provision for multi-processor communication. In these modes, 9 data bits are received. The 9th bit goes into RB8. Then a stop bit follows. The port can be programmed such that when the stop bit is received, the serial port 0 interrupt will be activated (i.e. the request flag RI is set) only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor communications is as follows.

If the master processor wants to transmit a block of data to one of the 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. After having received a complete message, the slave sets SM2 again. The slaves that were not addressed leave their SM2 set and go on about their business, ignoring the incoming data bytes.

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

7.2.3 Baud Rates

As already mentioned there are several possibilities to generate the baud rate clock for the serial interface depending on the mode in which it is operated.

To clarify the terminology, something should be said about the difference between "baud rate clock" and "baud rate". The serial interface requires a clock rate which is 16 times the baud rate for internal synchronization, as mentioned in the detailed description of the various operating modes in section 7.2.4.

Therefore, the baud rate generator have to provide a "baud rate clock" to the serial interface which - there divided by 16 - results in the actual "baud rate". However, all formulas given in the following section already include the factor and calculate the final baud rate.

Mode 0

The baud rate in mode 0 is fixed:

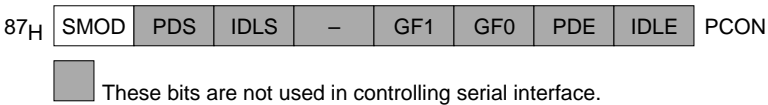
$$\text{Mode 0 baud rate} = \frac{\text{oscillator frequency}}{12}$$

Mode 2

The baud rate in mode 2 depends on the value of bit SMOD in special function register PCON (see **figure 7-9**). If SMOD = 0 (which is the value after reset), the baud rate is 1/64 of the oscillator frequency. If SMOD = 1, the baud rate is 1/32 of the oscillator frequency.

$$\text{Mode 2 baud rate} = \frac{2^{\text{SMOD}}}{64} \times \text{oscillator frequency}$$

Figure 7-9
Special Function Register PCON (Address 87H)



Bit	Function
SMOD	When set, the baud rate of serial interface in modes 1, 2, 3 is doubled.

Modes 1 and 3

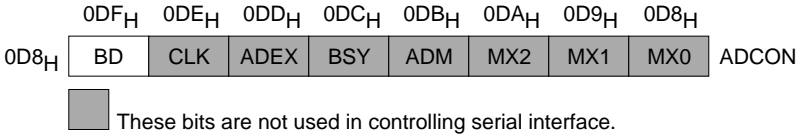
In these modes the baud rate is variable and can be generated alternatively by a dedicated baud rate generator or by timer 1.

Using the baud rate generator:

In modes 1 and 3, the SAB 80(C)515 can use the internal baud rate generator for the serial interface. To enable this feature, bit BD (bit 7 of special function register ADCON) must be set (see **figure 7-10**). This baud rate generator divides the oscillator frequency by 2500. Bit SMOD (PCON.7) also can be used to enable a multiply by two prescaler (see **figure 7-9**). At 12-MHz oscillator frequency, the commonly used baud rates 4800 baud (SMOD = 0) and 9600 baud (SMOD = 1) are available. The baud rate is determined by SMOD and the oscillator frequency as follows:

$$\text{Mode 1, 3 baud rate} = \frac{2^{\text{SMOD}}}{2500} \times \text{oscillator frequency}$$

Figure 7-10
Special Function Register ADCON (Address 0D8_H)



Bit	Function
BD	Baud rate enable. When set, the baud rate in modes 1 and 3 of serial interface is taken from a dedicated prescaler. Standard baud rates 4800 and 9600 baud at 12-MHz oscillator frequency can be achieved.

Using timer 1 to generate baud rates:

Timer 1 can be used for generating baud rates in mode 1 and 3 of the serial channel. Then the baud rate is determined by the timer 1 overflow rate and the value of SMOD as follows:

$$\text{Mode 1, 3 baud rate} = \frac{2^{\text{SMOD}}}{32} \times (\text{Timer 1 OV-rate})$$

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

$$\text{Mode 1, 3 baud rate} = \frac{2^{\text{SMOD}} \times \text{oscillator frequency}}{32 \times 12 \times (256 - (\text{TH1}))}$$

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

Table 7-4 lists various commonly used baud rates and shows how they can be obtained from timer 1.

Table 7-4
Timer 1 Generated Commonly Used Baud Rates

Baud Rate	f_{osc} (MHz)	SMOD	Timer 1			
			C/T	Mode	Reload Value	
Mode 1, 3:	62.5 Kbaud	12.0	1	0	2	FF _H
	19.5 Kbaud	11.059	1	0	2	FD _H
	9.6 Kbaud	11.059	0	0	2	FD _H
	4.8 Kbaud	11.059	0	0	2	FA _H
	2.4 Kbaud	11.059	0	0	2	F4 _H
	1.2 Kbaud	11.059	0	0	2	E8 _H
	110 Baud	6.0	0	0	2	72 _H
110 Baud	12.0	0	0	1	FE5B _H	

Figure 7-11 shows the mechanisms for baud rate generation of serial channel, while table 7-5 summarizes the baud rate formulas for all usual configurations.

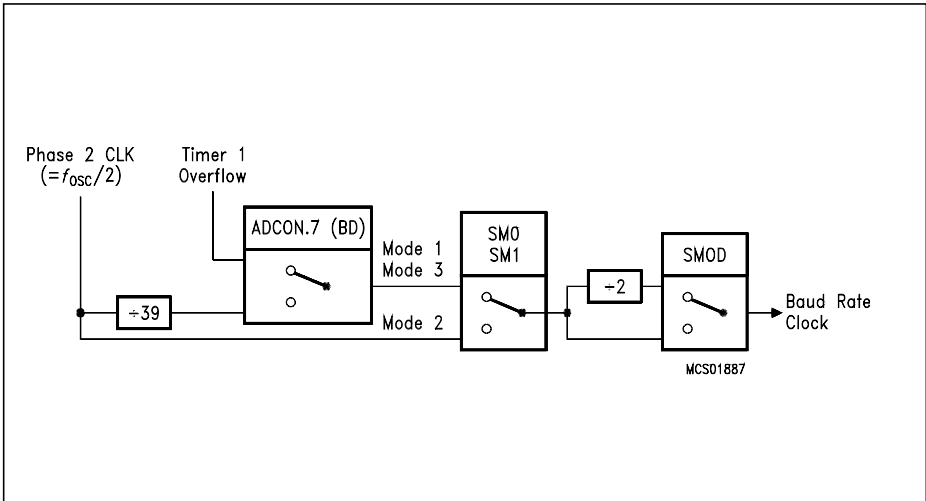


Figure 7-11
Generation of Baud Rates for Serial Interface

Table 7-5
Baud Rates of Serial Interface 0

Baud Rate Derived from	Interface Mode	Baud Rate
Timer 1 in mode 1	1, 3	$\frac{2^{SMOD}}{2} \times \frac{1}{16} \times (\text{timer 1 overflow rate})$
Timer 1 in mode 2	1, 3	$\frac{2^{SMOD}}{2} \times \frac{1}{16} \times \frac{f_{OSC}}{12 \times (256 - (TH1))}$
Oscillator	2	$\frac{2^{SMOD}}{2} \times \frac{1}{16} \times \frac{f_{OSC}}{2}$
Baud rate generator	1, 3	$\frac{2^{SMOD}}{2} \times \frac{1}{16} \times \frac{f_{OSC}}{1250}$

7.2.4 Detailed Description of the Operating Modes

The following sections give a more detailed description of the several operating modes of the serial interface.

7.2.4.1 Mode 0, Synchronous Mode

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 of the oscillator frequency.

Figures 7-16 a) and b) show 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 P3.0, and also enables SHIFT CLOCK to the alternate output function line P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1, and S2, while the interface is transmitting. Before and after transmission SHIFT CLOCK remains high. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register is shifted one position to the right.

As data bits shift 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 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 deactivates SEND and sets TI. Both of these actions occur at S1P1 in the 10th machine cycle after "write-to-SBUF".

Reception is initiated by the condition REN = 1 and RI = 0. At S6P2 in the next machine cycle, the RX control unit writes the bits 1111 1110 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 in every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted one position to the left. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 in the same machine cycle.

As data bits come in from the right, 1 s 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 in the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

7.2.4.2 Mode 1, 8-Bit UART

Ten bits are transmitted (through TxD), or received (through RxD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On reception through RxD, the stop bit goes into RB8 (SCON).

The baud rate for serial interface 0 is determined by the timer 1 overflow rate or by the internal baud rate generator.

Figures 7-17 a) and b) show a simplified functional diagram of the serial channel in mode 1. The generation of the baud rate clock is described in section 7.2.3.

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 block that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next roll-over 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 to 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 left of the MSB, and all positions to the left of that contain zero. This condition flags the TX control to do one last shift and then deactivates $\overline{\text{SEND}}$ and sets 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 reception 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 rollover with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16 counter states. At the 7th, 8th and 9th counter state 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 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 from the right, 1's 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. 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, no matter whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RxD.

7.2.4.3 Mode 2, 9-Bit UART

Mode 2 is functionally identical to mode 3 (see below). The only exception is, that in mode 2 the baud rate can be programmed to two fixed quantities: either 1/32 or 1/64 of the oscillator frequency. In mode 3 the baud rate clock is generated by timer 1, which is incremented by a rate of $f_{osc}/12$ or by the internal baud rate generator.

7.2.4.4 Mode 3, 9-Bit UART

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 transmission, the 9th data bit (TB8) can be assigned the value of 0 or 1. On reception the 9th data bit goes into RB8 in SCON.

The baud rate is generated by either using timer 1 or the internal baud rate generator (see section 7.2.3).

Figures 7-18 a) and b) show a functional diagram of the serial interfaces in mode 2 and 3 and associated timing. 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, and not to the "write-to-SBUF" signal).

The transmission begins with the activation of \overline{SEND} , which puts the start bit to TxD. One bit time later, DATA is activated which enables the output bit of 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 zeros are clocked in. Thus, as data shift out to the right, zeros are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just left of the TB8, and all positions to the left of that contain zeros.

This condition flags the TX control unit to do one last shift and then deactivate \overline{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 1FH is written to the input shift register.

At the 7th, 8th and 9th counter state 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. 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 from the right, 1 s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which 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 9th data bit = 1

If either one of these two 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, the first 8 data bits go into SBUF. One bit time later, no matter whether the above conditions are met or not, the unit goes back to look for a 1-to-0 transition at the RxD.

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

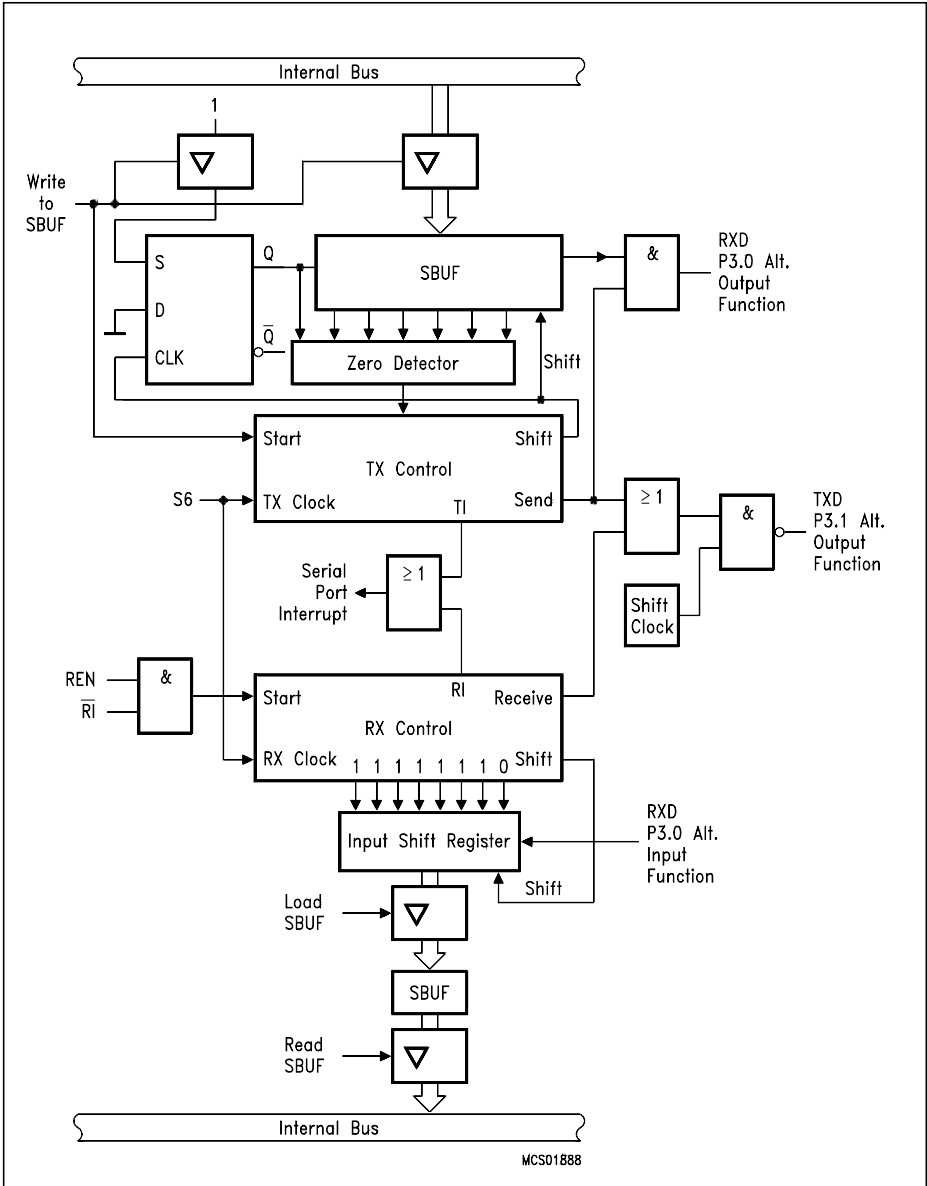


Figure 7-16 a)
Functional Diagram - Serial Interface, Mode 0

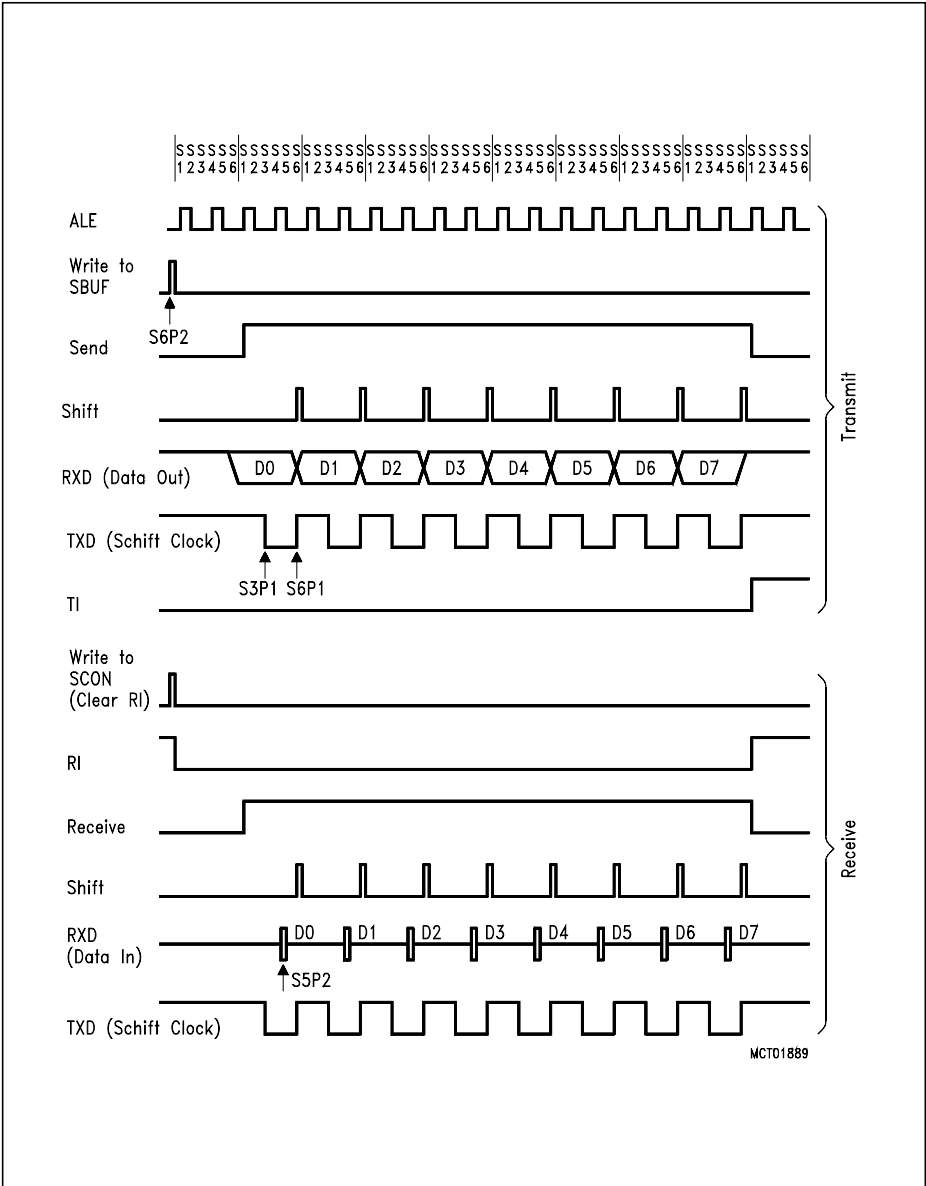


Figure 7-16 b)
Timing Diagram - Serial Interface, Mode 0

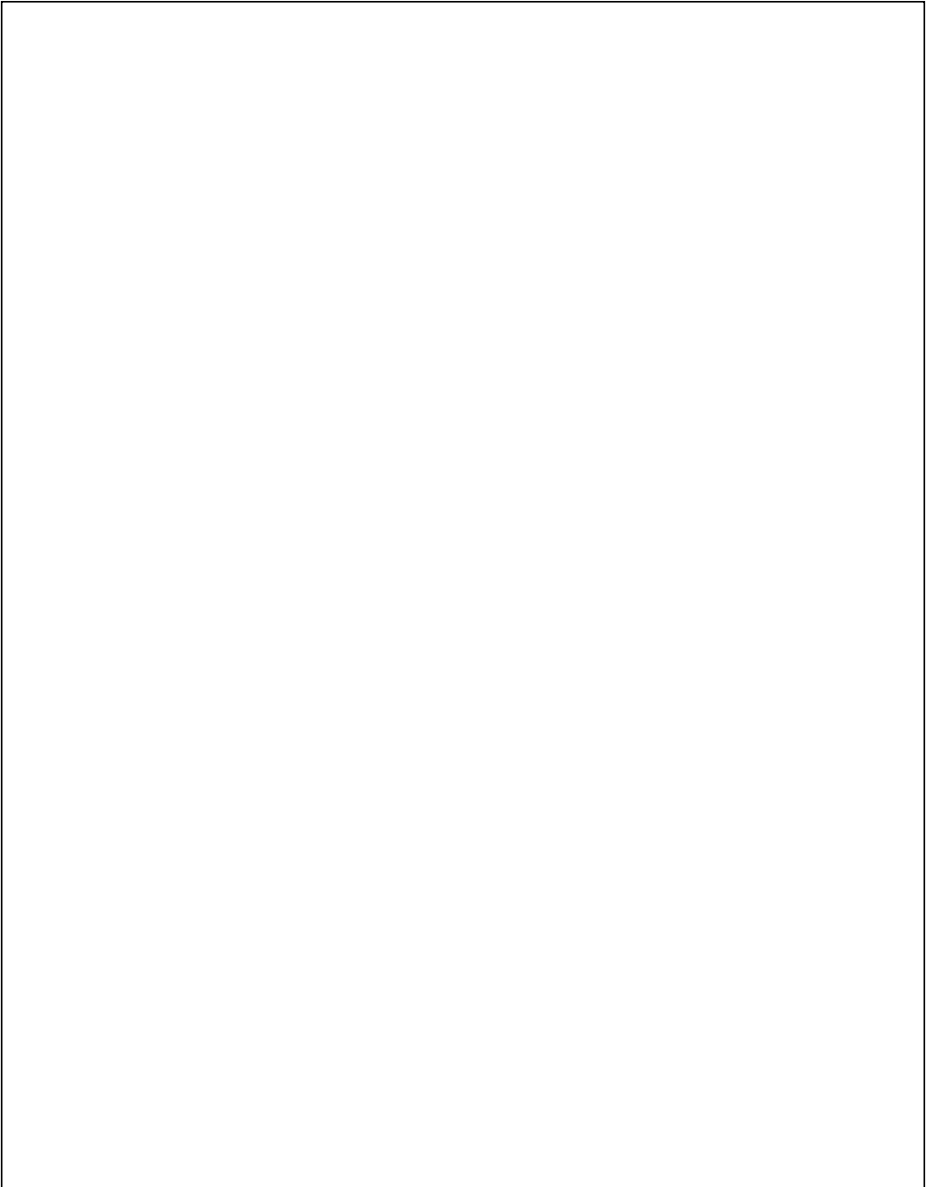


Figure 7-17 a)
Functional Diagram - Serial Interface, Mode 1

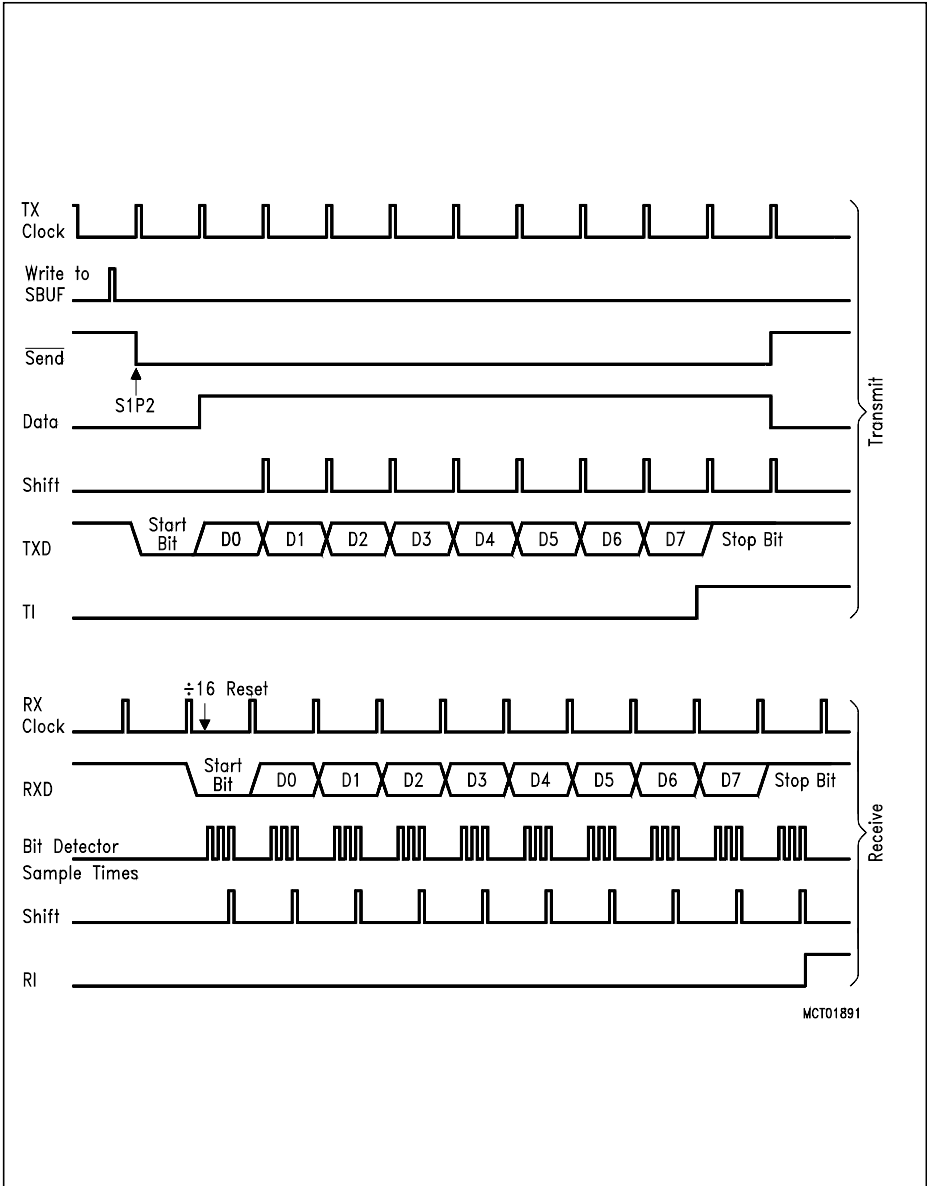


Figure 7-17 b) Timing Diagram - Serial Interface, Mode 1

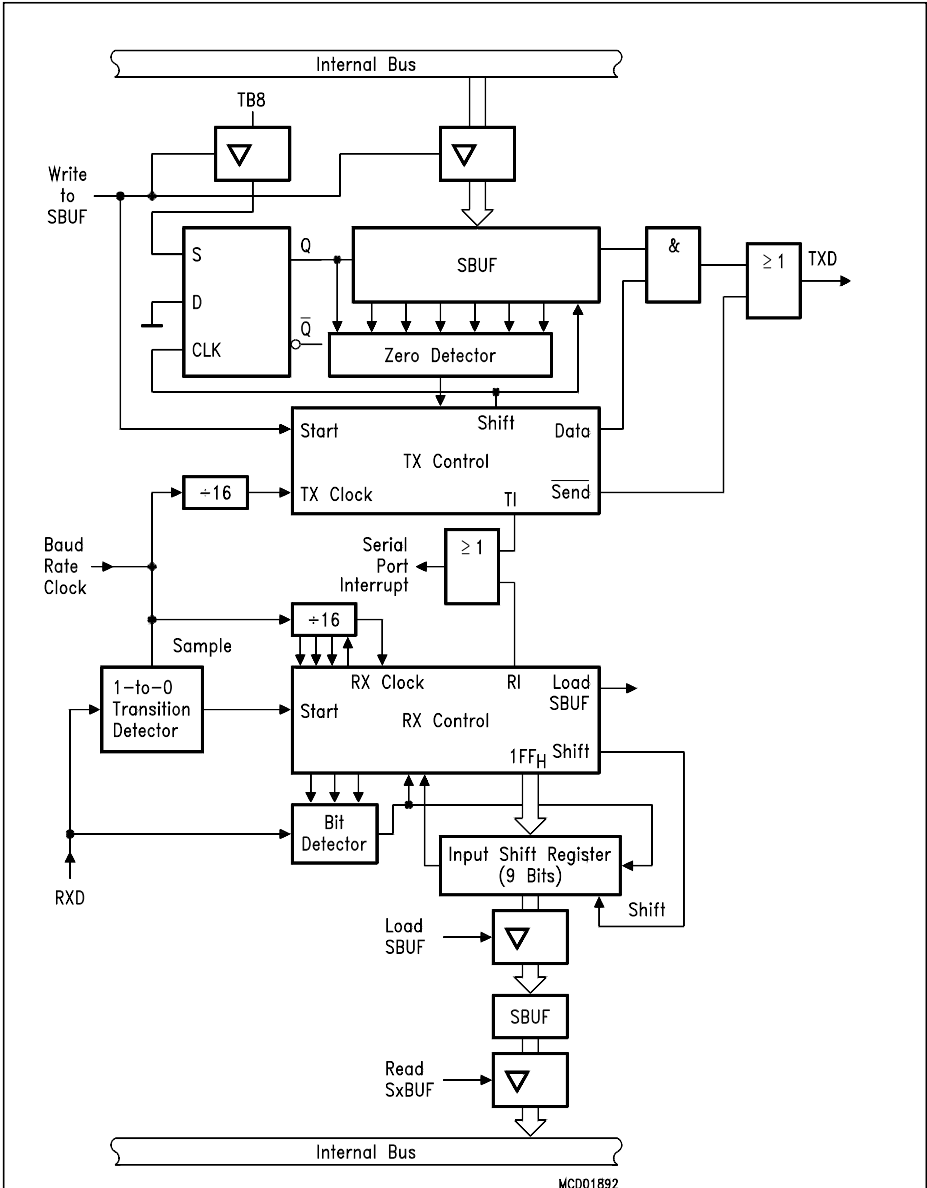


Figure 7-18 a)
Functional Diagram - Serial Interface, Modes 2 and 3

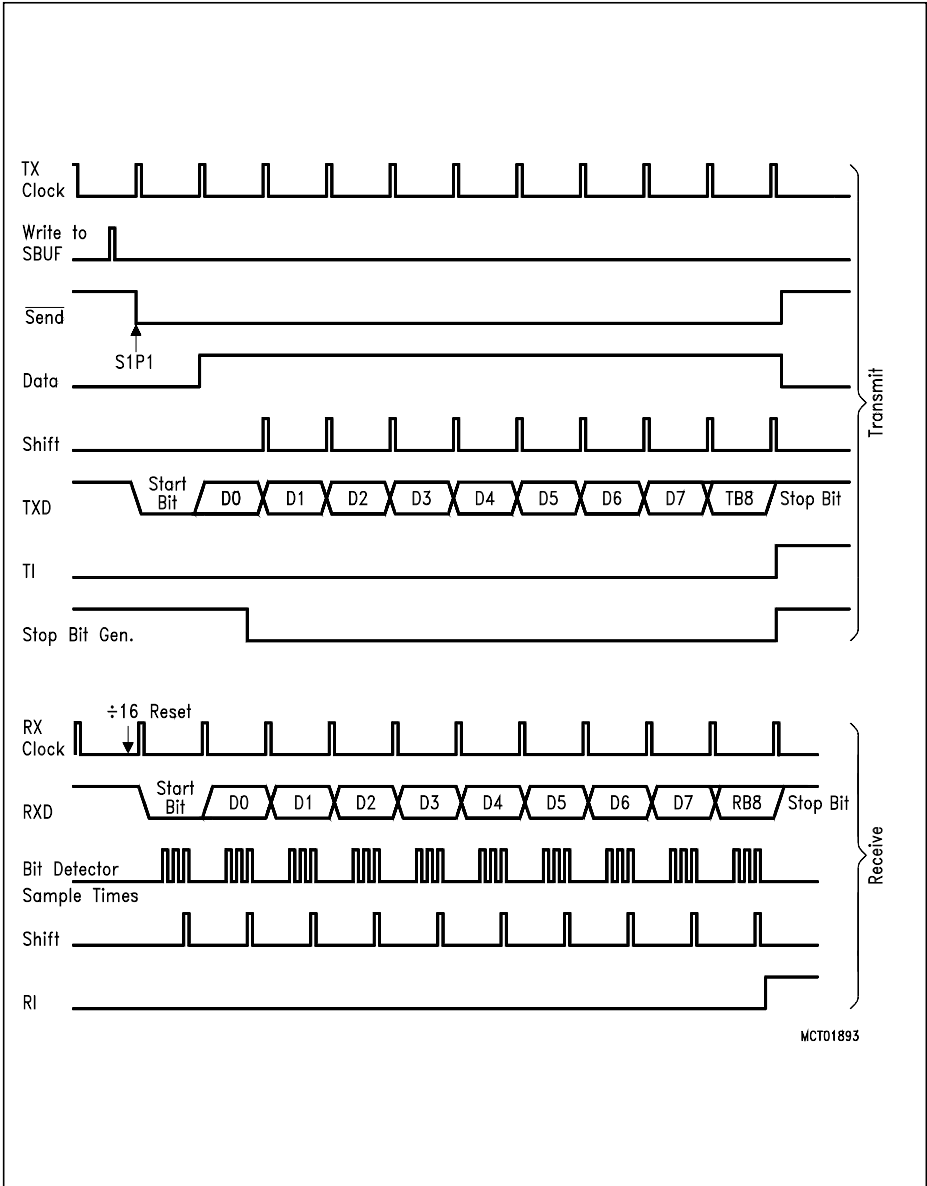


Figure 7-18 b)
Timing Diagram - Serial Interface, Modes 2 and 3

7.3 Timer 0 and Timer 1

The SAB 80(C)515 three general purpose 16-bit timer/counters: timer 0, timer 1, timer 2 and the compare timer (timer 2 is discussed separately in section 7.5). Timer/counter 0 and 1 are fully compatible with timer/counters 0 and 1 of the SAB 80(C)51 and can be used in the same operating modes.

Timer/counter 0 and 1 which are discussed in this section can be configured to operate either as timers or event counters:

- In "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 "counter" function, the register is incremented in response to a 1-to-0 transition (falling edge) at its corresponding external input pin, T0 or T1 (alternate functions of P3.4 and P3.5, resp.). 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 two 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 must be held for at least one full machine cycle.

In addition to the "timer" and "counter" selection, timer/counters 0 and 1 have four operating modes from which to select.

Figure 7-19
Special Function Register TCON (Address 88H)

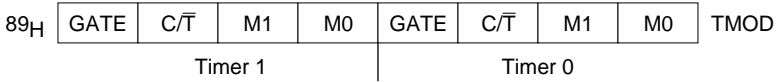


Bit	Function
TR0	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 ON/OFF.
TF0	Timer 0 overflow flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.
TR1	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 ON/OFF.
TF1	Timer 1 overflow flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.

Each timer consists of two 8-bit registers (TH0 and TL0 for timer/counter 0, TH1 and TL1 for timer/counter 1) which may be combined to one timer configuration depending on the mode that is established. The functions of the timers are controlled by two special function registers TCON and TMOD, shown in **figures 7-19** and **7-20**.

In the following descriptions the symbols TH0 and TL0 are used specify the high-byte and low-byte of timer 0 (TH1 and TL1 for timer 1, respectively). The operating modes are described and shown for timer 0. If not explicitly noted, this applies also to timer 1.

Figure 7-20
Special Function Register TMOD (Address 89H)



Timer/counter 0/1 mode control register.

Bit		Symbol
Gate		Gating control. When set, timer/counter "x" is enabled only while " \overline{INTx} " pin is high and "TRx" control bit is set. When cleared timer "x" is enabled whenever "TRx" control bit is set.
C/\overline{T}		Counter or timer select bit. Set for counter operation (input from "Tx" input pin). Cleared for timer operation (input from internal system clock).
M1	M0	
0	0	8-bit timer/counter "THx" operates as 8-bit timer/counter "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" holds 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. TH00 is an 8-bit timer only controlled by timer 1 control bits.
1	1	Timer 1: Timer/counter 1 stops

7.3.1 Mode 0

Putting either timer/counter into mode 0 configures it as an 8-bit timer/counter with a divide-by-32 prescaler. **Figure 7-21** shows the mode 0 operation.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1's to all 0's, it sets the timer overflow flag TFO. The overflow flag TFO then can be used to request an interrupt (see section 8 for details about the interrupt structure).

The counted input is enabled to the timer when TR0 = 1 and either GATE = 0 or $\overline{\text{INT0}} = 1$ (setting GATE = 1 allows the timer to be controlled by external input $\overline{\text{INT0}}$, to facilitate pulse width measurements). TR0 is a control bit in the special function register TCON; GATE is in TMOD.

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL0. The upper 3 bits of TL0 are indeterminate and should be ignored. Setting the run flag (TR0) does not clear the registers.

Mode 0 operation is the same for timer 0 as for timer 1. Substitute TR1, TF1, TH1, TL1, and $\overline{\text{INT1}}$ for the corresponding timer 1 signals in **figure 7-21**. There are two different gate bits, one for timer 1 (TMOD.7) and one for timer 0 (TMOD.3).

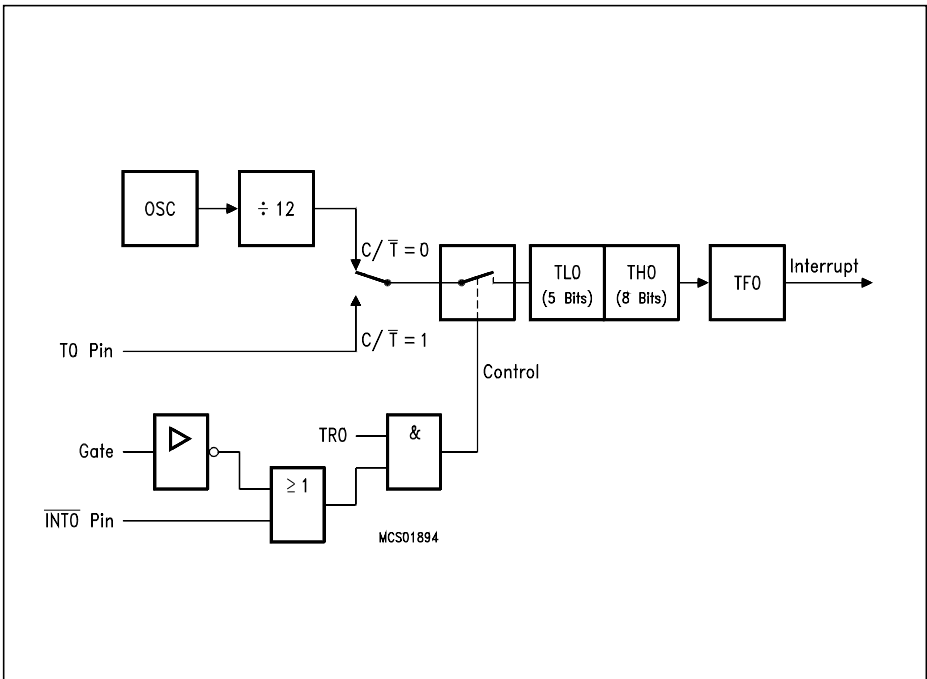


Figure 7-21
Timer/Counter 0, Mode 0: 13 Bit-Timer/Counter
 The same applies to timer/counter 1

7.3.2 Mode 1

Mode 1 is the same as mode 0, except that the timer register is run with all 16 bits. Mode 1 is shown in figure 7-22.

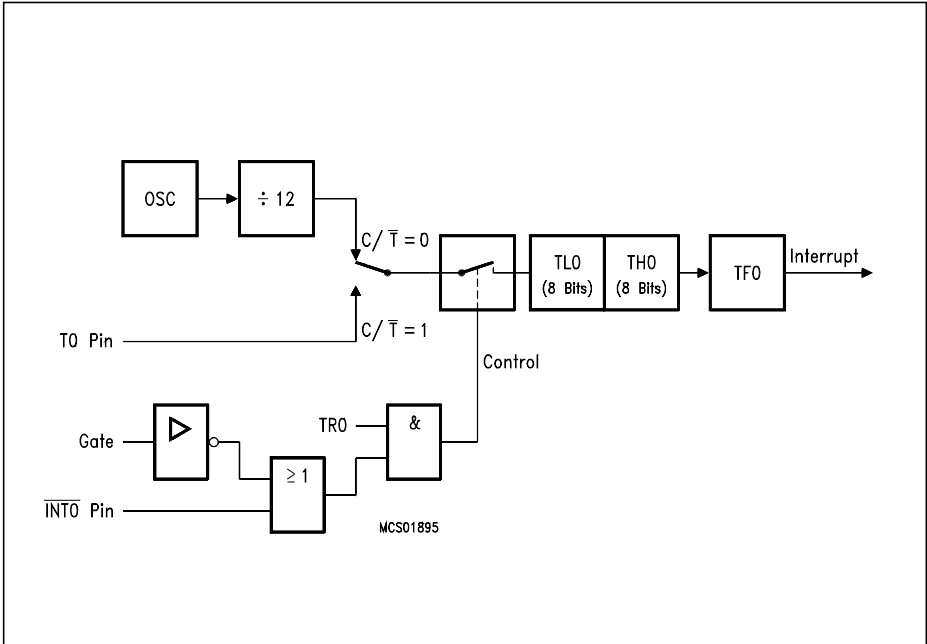


Figure 7-22
Timer/Counter 0, Mode 1: 16-Bit Timer/Counter
The same applies to timer/counter 1

7.3.3 Mode 2

Mode 2 configures the timer register as an 8-bit counter (TLO) with automatic reload, as shown in figure 7-23. Overflow from TLO not only sets TFO, but also reloads TLO with the contents of TH0, which is preset by software. The reload leaves TH0 unchanged.

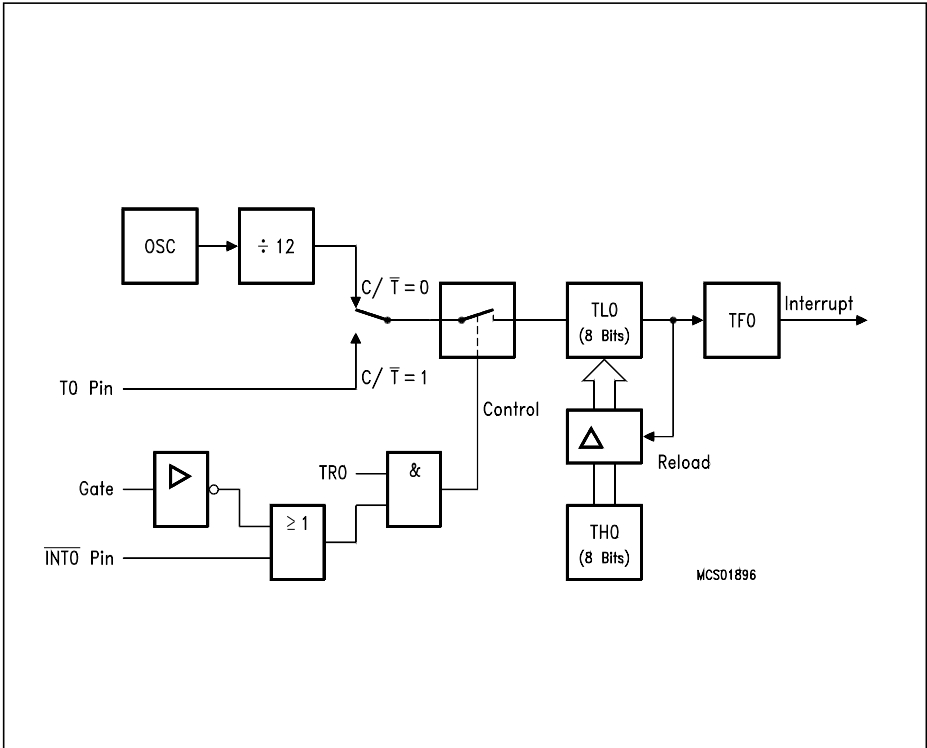


Figure 7-23
Timer/Counter 0, Mode 2: 8-Bit Timer/Counter with Auto-Reload
 The same applies to timer/counter 1

7.3.4 Mode 3

Mode 3 has different effects on timer 0 and timer 1. Timer 1 in mode 3 simply holds its count. The effect is the same as setting TR1 = 0. 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 7-24**. TL0 uses the timer 0 control bits: C/T, 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. 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 channel as a baud rate generator, or in fact, in any application not requiring an interrupt from timer 1 itself.

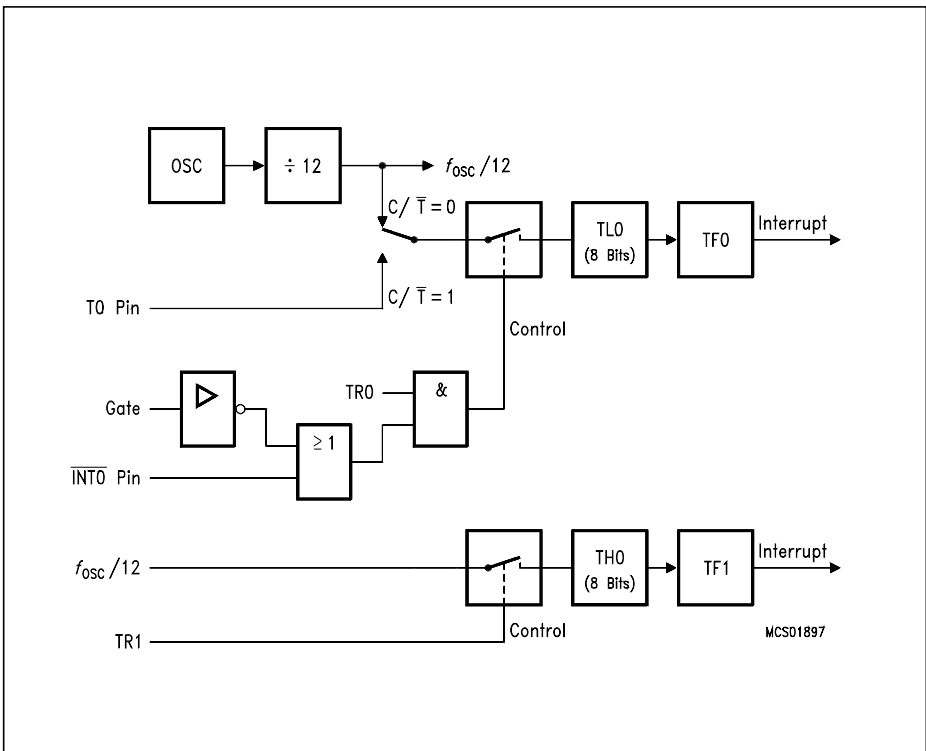


Figure 7-24
Timer/Counter 0, Mode 3: Two 8-Bit Timers/Counters

7.4 A/D Converter

The SAB 80(C)515 provides an A/D converter with the following features:

- Eight multiplexed input channels
- The possibility of using the analog input channels (port 6) as digital inputs (ACMOS version only).
- Programmable internal reference voltages (16 steps each) via resistor array
- 8-bit resolution within the selected reference voltage range
- 13 machine cycles conversion time for ACMOS versions (including sample time)
- 15 machine cycles conversion time for MYMOS versions (including sample time)
- Internal start-of-conversion trigger
- Interrupt request generation after each conversion

For the conversion, the method of successive approximation via capacitor array is used. The externally applied reference voltage range has to be held on a fixed value within the specifications (see section "A/D Converter Characteristics" in the data sheet). The internal reference voltages can be varied to reduce the reference voltage range of the A/D converter and thus to achieve a higher resolution.

Figure 7-25 shows a block diagram of the A/D converter. There are three user-accessible special function registers:

ADCON (A/D converter control register), ADDAT (A/D converter data register) and DAPR (D/A converter program register) for the programmable reference voltages.

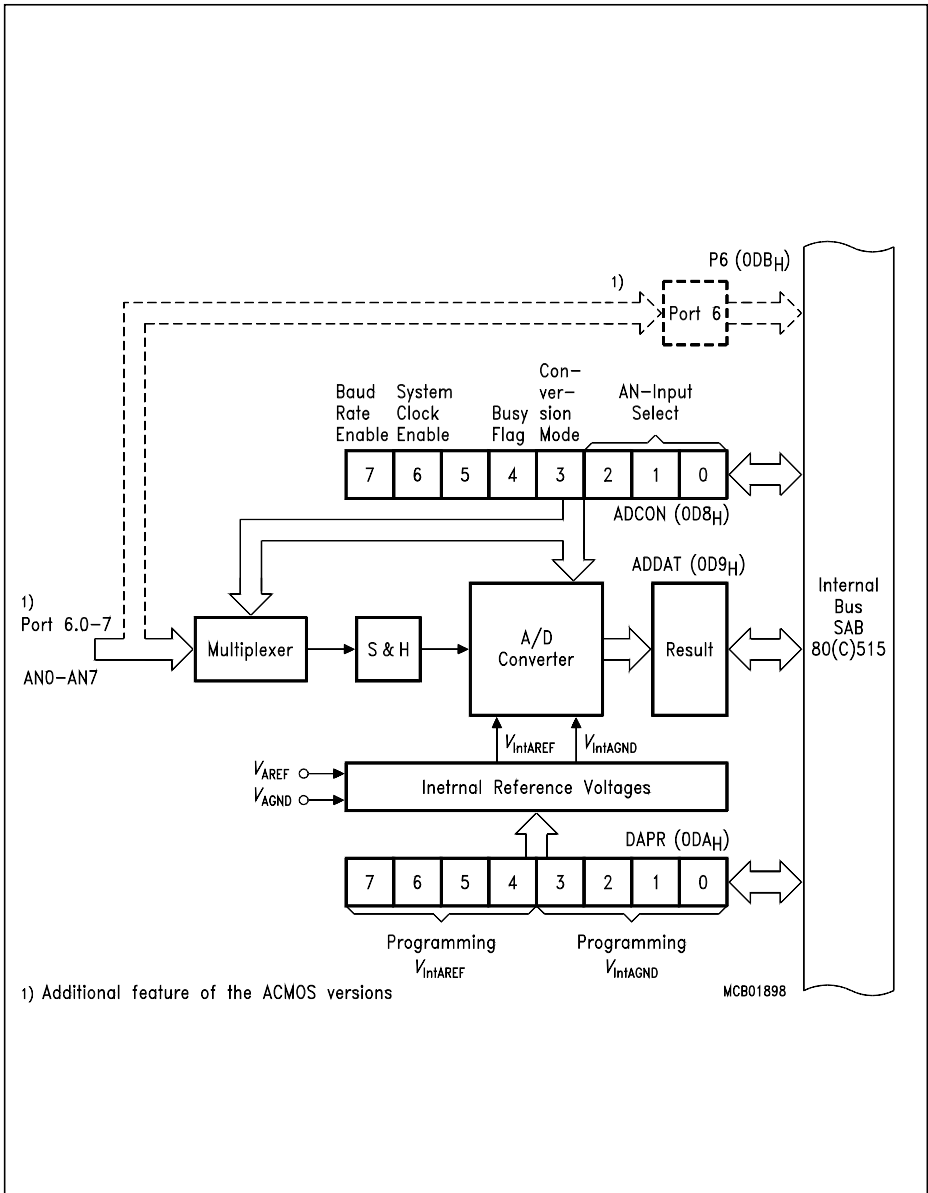


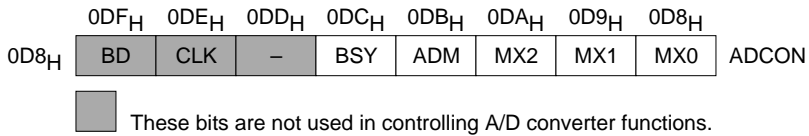
Figure 7-25
A/D Converter Block Diagram

7.4.1 Function and Control

7.4.1.1 Initialization and Input Channel Selection

Special function register ADCON which is illustrated in **figure 7-26** is used to set the operating modes, to check the status, and to select one of eight analog input channels.

Figure 7-26
Special Function Register ADCON (Address 0D8_H)



Bit	Function
MX0 MX1 MX2	Select 8 input channels of the A/D converter, see table 7-6
ADM	A/D conversion mode. When set, a continuous conversion is selected. If ADM = 0, the converter stops after one conversion.
BSY	Busy flag. This flag indicates whether a conversion is in progress (BSY = 1). The flag is cleared by hardware when the conversion is completed.

Register ADCON contains two mode bits. Bit ADM is used to choose the single or continuous conversion mode. In single conversion mode only one conversion is performed after starting, while in continuous conversion mode after the first start a new conversion is automatically started on completion of the previous one.

The busy flag BSY (ADCON.4) is automatically set when a conversion is in progress. After completion of the conversion it is reset by hardware. This flag can be read only, a write has no effect. There is also an interrupt request flag IADC (IRCON.0) that is set when a conversion is completed. See section 8 for more details about the interrupt structure.

Table 7-6
Selection of the Analog Input Channels

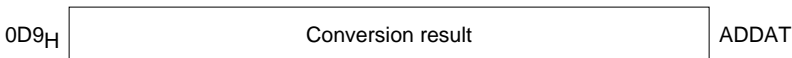
MX2	MX1	MX0	Selected Channel	Pin	
				MYMOS	ACMOS
0	0	0	Analog input 0	AN0	P6.0
0	0	1	Analog input 1	AN1	P6.1
0	1	0	Analog input 2	AN2	P6.2
0	1	1	Analog input 3	AN3	P6.3
1	0	0	Analog input 4	AN4	P6.4
1	0	1	Analog input 5	AN5	P6.5
1	1	0	Analog input 6	AN6	P6.6
1	1	1	Analog input 7	AN7	P6.7

The bits MX0 to MX2 in special function register ADCON are used for selection of the analog input channels.

Port 6 of the ACMOS versions is a dual purpose input port. If the input voltage meets the specified logic levels, it can be used as digital input as well regardless of whether the pin levels are sampled by the A/D converter at the same time.

The special function register ADDAT (**figure 7-28**) holds the converted digital 8-bit data result. The data remains in ADDAT until it is overwritten by the next converted data. ADDAT can be read or written under software control. If the A/D converter of the SAB 80(C)515 is not used, register ADDAT can be used as an additional general purpose register.

Figure 7-28
Special Function Register ADDAT (Address 0D9H)



This register contains the 8-bit conversion result.

7.4.1.2 Start of Conversion

An internal start of conversion is triggered by a write-to-DAPR instruction. The start procedure itself is independent of the value which is written to DAPR. However, the value in DAPR determines which internal reference voltages are used for the conversion (see section 7.4.2).

When single conversion mode is selected (ADM = 0) only one conversion is performed. In continuous mode after completion of a conversion a new conversion is triggered automatically, until bit ADM is reset.

7.4.2 Reference Voltages

The SAB 80(C)515 has two pins to which a reference voltage range for the on-chip A/D converter is applied (pin V_{AREF} for the upper voltage and pin V_{AGND} for the lower voltage). In contrast to conventional A/D converters it is now possible to use not only these externally applied reference voltages for the conversion but also internally generated reference voltages which are derived from the externally applied ones. For this purpose a resistor ladder provides 16 equidistant voltage levels between V_{AREF} and V_{AGND} . These steps can individually be assigned as upper and lower reference voltage for the converter itself. These internally generated reference voltages are called V_{IntAREF} and V_{IntAGND} . The internal reference voltage programming can be thought of as a programmable "D/A converter" which provides the voltages V_{IntAREF} and V_{IntAGND} for the A/D converter itself.

The SFR DAPR (see figure 7-29) is provided for programming the internal reference voltages V_{IntAREF} and V_{IntAGND} . For this purpose the internal reference voltages can be programmed in steps of 1/16 of the external reference voltages ($V_{\text{AREF}} - V_{\text{AGND}}$) by four bits each in register DAPR. Bits 0 to 3 specify V_{IntAGND} , while bits 4 to 7 specify V_{IntAREF} . A minimum of 1 V difference is required between the internal reference voltages V_{IntAREF} and V_{IntAGND} for proper operation of the A/D converter. This means, for example, in the case where V_{AREF} is 5 V and V_{AGND} is 0 V, there must be at least four steps difference between the internal reference voltages V_{IntAREF} and V_{IntAGND} .

The values of V_{IntAGND} and V_{IntAREF} are given by the formulas:

$$V_{\text{IntAGND}} = V_{\text{AGND}} + \frac{\text{DAPR} (.3-.0)}{16} (V_{\text{AREF}} - V_{\text{AGND}})$$

with $\text{DAPR} (.3-.0) < C_H$;

$$V_{\text{IntAREF}} = V_{\text{AGND}} + \frac{\text{DAPR} (.7-.4)}{16} (V_{\text{AREF}} - V_{\text{AGND}})$$

with $\text{DAPR} (.7-.4) > 3C_H$;

$\text{DAPR} (.3-.0)$ is the contents of the low-order nibble, and $\text{DAPR} (.7-.4)$ the contents of the high-order nibble of DAPR.

If DAPR (.3-.0) or DAPR (.7-.4) = 0, the internal reference voltages correspond to the external reference voltages V_{AGND} and V_{AREF} , respectively.

If $V_{AINPUT} > V_{IntAREF}$, the conversion result is 0FF_H, if $V_{AINPUT} < V_{IntAGND}$, the conversion result is 00_H (V_{AINPUT} is the analog input voltage).

If the external reference voltages $V_{AGND} = 0$ V and $V_{AREF} = +5$ V (with respect to V_{SS} and V_{CC}) are applied, then the following internal reference voltages $V_{IntAGND}$ and $V_{IntAREF}$ shown in **table 7-7** can be adjusted via the special function register DAPR.

Figure 7-29
Special Function Register DAPR (Address DA_H)



D/A converter program register. Each 4-bit nibble is used to program the internal reference voltages. Write-access to DAPR starts conversion.

$$V_{IntAGND} = V_{AGND} + \frac{\text{DAPR (.3-.0)}}{16} (V_{AREF} - V_{AGND})$$

with DAPR (.3-.0) < 13;

$$V_{IntAGND} = V_{AGND} + \frac{\text{DAPR (.7-.4)}}{16} (V_{AREF} - V_{AGND})$$

with DAPR (.7-.4) > 3;

Table 7-7
Adjustable Internal Reference Voltages

Step	DAPR (.3-.0) DAPR (.7-.4)	$V_{IntAGND}$	$V_{IntAREF}$
0	0000	0.0	5.0
1	0001	0.3125	–
2	0010	0.625	–
3	0011	0.9375	–
4	0100	1.25	1.25
5	0101	1.5625	1.5625
6	0110	1.875	1.875
7	0111	2.1875	2.1875
8	1000	2.5	2.5
9	1001	2.8125	2.8125
10	1010	3.125	3.125
11	1011	3.4375	3.4375
12	1100	3.75	3.75
13	1101	–	4.0625
14	1110	–	4.375
15	1111	–	4.68754

The programmability of the internal reference voltages allows adjusting the internal voltage range to the range of the external analog input voltage or it may be used to increase the resolution of the converted analog input voltage by starting a second conversion with a compressed internal reference voltage range close to the previously measured analog value. **Figures 7-30** and **7-31** illustrate these applications.

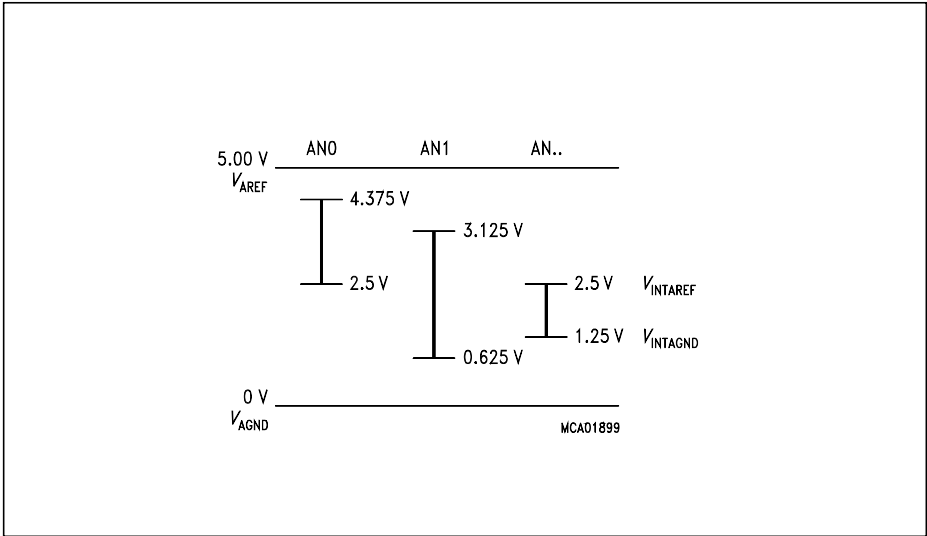


Figure 7-30
Adjusting the Internal Reference Voltages within Range of the External Analog Input Voltages

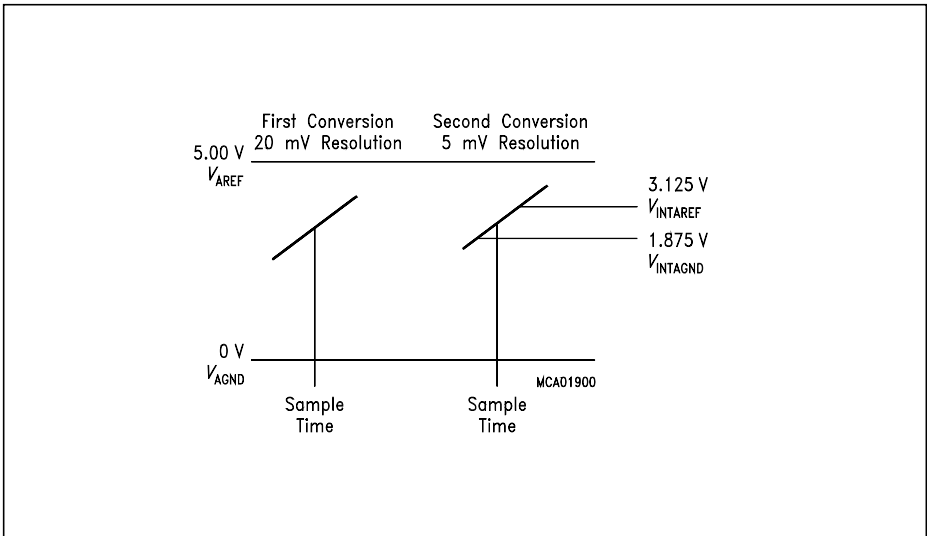


Figure 7-31
Increasing the Resolution by a Second Conversion

The external reference voltage supply need only be applied when the A/D converter is used, otherwise the pins V_{AREF} and V_{AGND} may be left unconnected. The reference voltage supply has to meet some requirements concerning the level of V_{AGND} and V_{AREF} and the output impedance of the supply voltage (see also "A/D Converter Characteristics" in the data sheet).

- The voltage V_{AREF} must meet the following specification:
 $V_{AREF} = V_{CC} \pm 5\%$
- The voltage V_{AGND} must meet a similar specification:
 $V_{AGND} = V_{SS} \pm 0.2\text{ V}$
- The differential output impedance of the analog reference supply voltage should be less than $1\text{ k}\Omega$

If the above mentioned operating conditions are not met the accuracy of the converter may be decreased.

Furthermore, the analog input voltage V_{INPUT} must not exceed the range from ($V_{AGND} - 0.2\text{ V}$) to ($V_{AREF} + 0.2\text{ V}$). Otherwise, a static input current might result at the corresponding analog input which will also affect the accuracy of the other input channels.

7.4.3 A/D Converter Timing

A conversion is started by writing into special function register DAPR. A write-to-DAPR will start a new conversion even if a conversion is currently in progress. The conversion begins with the next machine cycle and the busy flag BSY will be set.

The conversion procedure is divided into three parts:

Load time (t_L):

During this time the analog input capacitance C_1 (see data sheet) must be loaded to the analog input voltage level. The external analog source needs to be strong enough to source the current to load the analog input capacitance during the load time. This causes some restrictions for the impedance of the analog source. For a typical application the value of the impedance should be less than approx. $5\text{ k}\Omega$.

Sample time (t_S):

During this time the internal capacitor array is connected to the selected analog input channel. The sample time includes the load time which is described above. After the load time has passed the selected analog input must be held constant for the rest of the sample time. Otherwise the internal calibration of the comparator circuitry could be affected which might result in a reduced accuracy of the converter. However, in typical applications a voltage change of approx. $200 - 300\text{ mV}$ at the inputs during this time has no effect.

Conversion time (t_c):

The conversion time t_c includes the sample and load time. Thus, t_c is the total time required for one conversion. After the load time and sample time have elapsed, the conversion itself is performed during the rest of t_c . In the last machine cycle the converted result is moved to ADDAT; the busy flag (BSY) is cleared before. The A/D converter interrupt is generated by bit IADC in register IRCON. IADC is already set some cycles before the result is written to ADDAT. The flag IADC is set before the result is available in ADDAT because the shortest possible interrupt latency time is taken into account in order to ensure optimal performance. Thus, the converted result appears at the same time in ADDAT when the first instruction of the interrupt service routine is executed. Similar considerations apply to the timing of the flag BSY where usually a "JB BSY,\$" instruction is used for polling.

If a continuous conversion is established, the next conversion is automatically started in the machine cycle following the last cycle of the previous conversion.

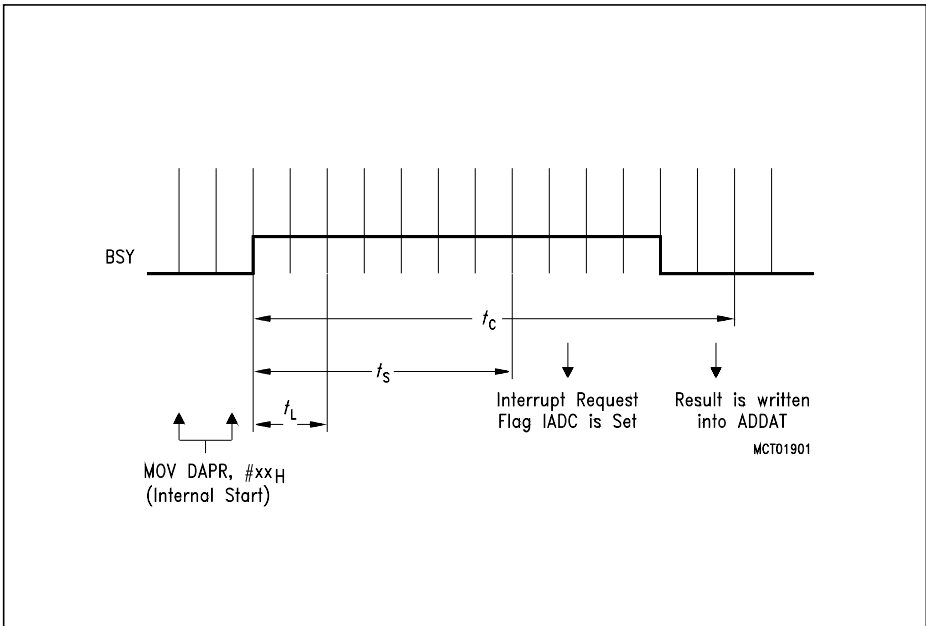


Figure 7-32
Timing Diagram of an A/D Converter

7.5 Timer 2 with Additional Compare/Capture/Reload

The timer 2 with additional compare/capture/reload features is one of the most powerful peripheral units of the SAB 80(C)515. It is used for all kinds of digital signal generation and event capturing like pulse generation, pulse width modulation, pulse width measuring etc.

This allows various automotive control applications (ignition/injection-control, anti-lock-brake ...) as well as industrial applications (DC-, three-phase AC- and stepper-motor control, frequency generation, digital-to-analog conversion, process control ...).

Please note that this timer is not equivalent to timer 2 of the SAB 80(C)52 (see section 7.5.1).

Timer 2 in combination with the compare/capture/reload registers allows the following modes:

- Compare: up to 4 PWM signals with 65535 steps at maximum, and 1 μ sec resolution
- Capture: up to 4 high speed inputs with 1 μ sec resolution
- Reload: modulation of timer 2 cycle time

The block diagram in **figure 7-33 a)** shows the general configuration of timer 2 with the additional compare/capture/reload registers.

The corresponding port functions are listed in **table 7-8**.

Table 7-9 shows the additional special function registers of timer 2.

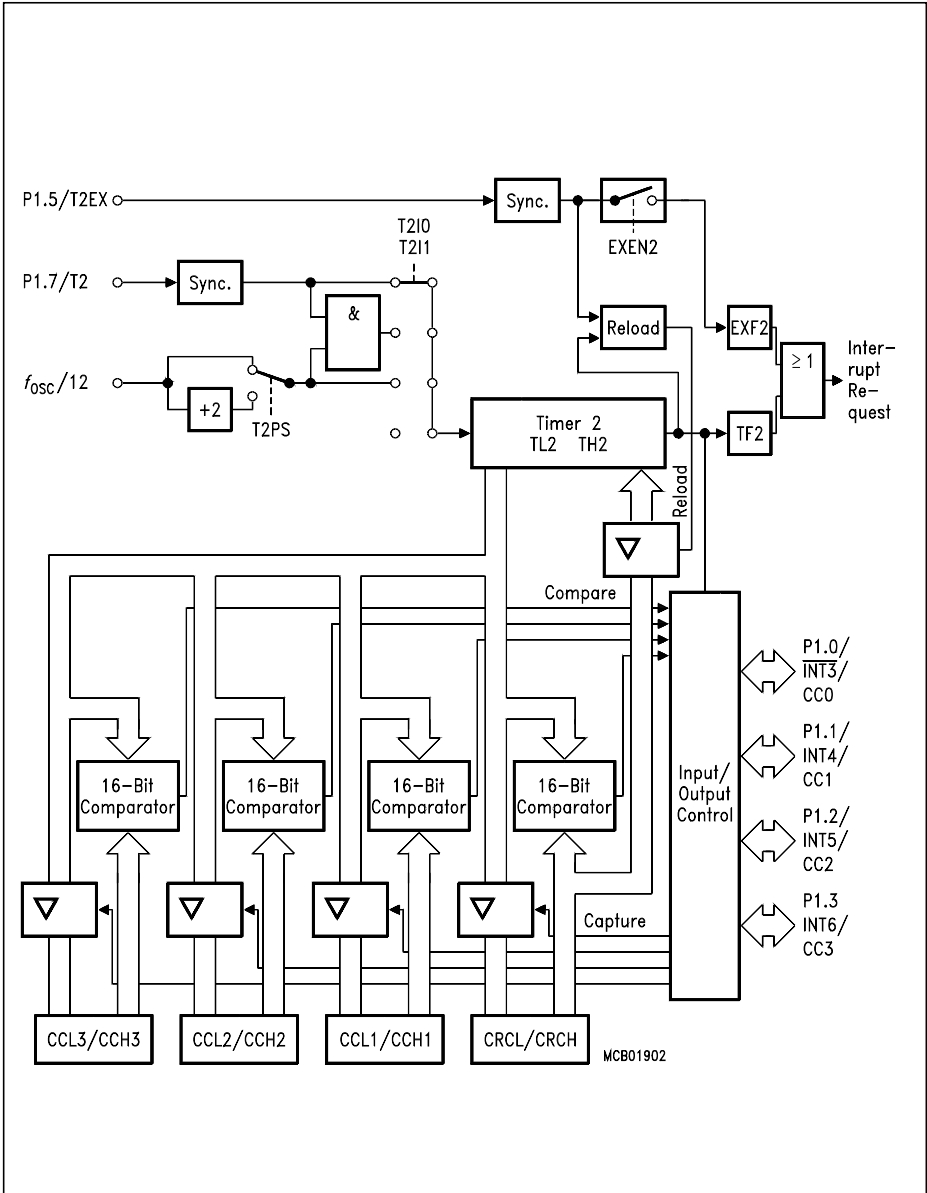


Figure 7-33 a)
Timer 2 Block Diagram

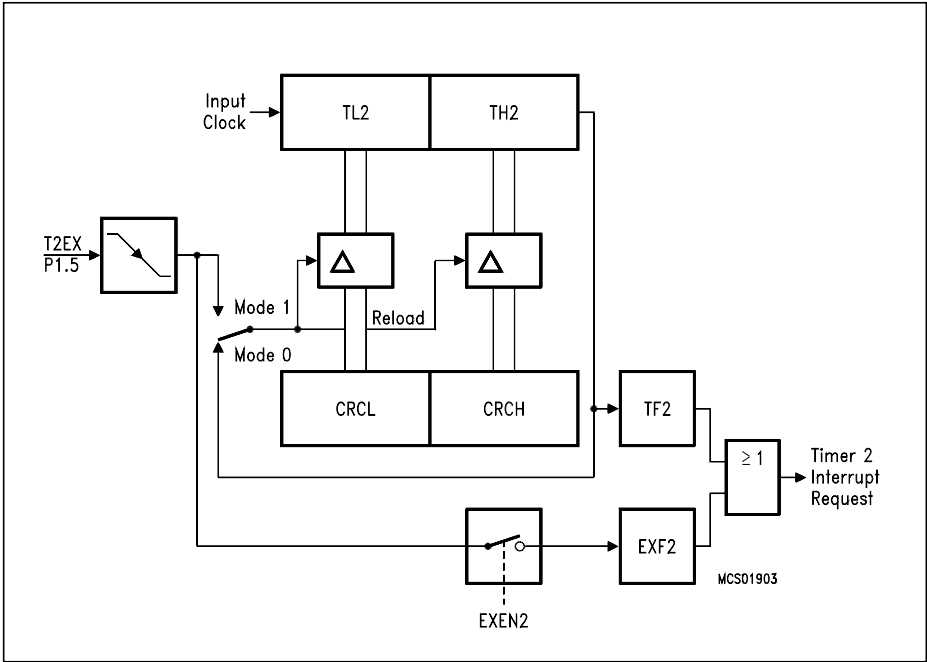


Figure 7-33 b)
Timer 2 in Reload Mode

Table 7-8
Alternate Port Functions of Timer 2

Pin Symbol	Input (I) Output (O)	Function
P1.7/T2	I/O	External count or gate input to timer 2
P1.5/T2EX	I/O	External reload trigger input
P1.3/INT6/CC3	I/O	Comp. output/capture input for CC register 3
P1.2/INT5/CC2	I/O	Comp. output/capture input for CC register 2
P1.1/INT4/CC1	I/O	Comp. output/capture input for CC register 1
P1.0/INT3/CC0	I/O	Comp. output/capture input for CRC register

Table 7-9
Additional Special Function Registers of Timer 2

Symbol	Description	Address
CCEN	Comp./capture enable reg.	0C1H
CCH1	Comp./capture reg. 1, high byte	0C3H
CCH2	Comp./capture reg. 2, high byte	0C5H
CCH3	Comp./capture reg. 3, high byte	0C7H
CCL1	Comp./capture reg. 1, low byte	0C2H
CCL2	Comp./capture reg. 2, low byte	0C4H
CCL3	Comp./capture reg. 3, low byte	0C6H
CRCH	Com./rel./capt. reg., high byte	0CBH
CRCL	Com./rel./capt. reg., low byte	0CAH
IRCON	Interrupt control register	0C0H
TH2	Timer 2, high byte	0CDH
TL2	Timer 2, low byte	0CCH
T2CON	Timer 2 control register	0C8H

7.5.1 Timer 2

The timer 2, which is a 16-bit-wide register, can operate as timer, event counter, or gated timer.

Timer Mode

In timer function, the count rate is derived from the oscillator frequency. A 2:1 prescaler offers the possibility of selecting a count rate of 1/12 or 1/24 of the oscillator frequency. Thus, the 16-bit timer register (consisting of TH2 and TL2) is either incremented in every machine cycle or in every second machine cycle. The prescaler is selected by bit T2PS in special function register T2CON (**see figure 7-35**). If T2PS is cleared, the input frequency is 1/12 of the oscillator frequency; if T2PS is set, the 2:1 prescaler gates 1/24 of the oscillator frequency to the timer.

Gated Timer Mode

In gated timer function, the external input pin T2 (P1.7) functions as a gate to the input of timer 2. If T2 is high, the internal clock input is gated to the timer. T2 = 0 stops the counting procedure. This will facilitate pulse width measurements. The external gate signal is sampled once every machine cycle (for the exact port timing, please refer to section 7.1 "Parallel I/O").

Event Counter Mode

In the counter function, the timer 2 is incremented in response to a 1-to-0 transition at its corresponding external input pin T2 (P1.7). In this function, the external input is sampled every machine cycle. When the sampled inputs show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the timer register in the cycle following the one in which the transition was detected. Since it takes two 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 must be held for at least one full machine cycle (see also section 7.1 "Parallel I/O" for the exact sample time at the port pin P1.7).

Note:

The prescaler must be off for proper counter operation of timer 2, i.e. T2PS must be 0.

In either case, no matter whether timer 2 is configured as timer, event counter, or gated timer, a rolling-over of the count from all 1's to all 0's sets the timer overflow flag TF2 (bit 6 in SFR IRCON, interrupt request control) which can generate an interrupt.

If TF2 is used to generate a timer overflow interrupt, the request flag must be cleared by the interrupt service routine as it could be necessary to check whether it was the TF2 flag or the external reload request flag EXF2 which requested the interrupt (for EXF2 see below). Both request flags cause the program to branch to the same vector address.

The input clock to timer 2 is selected by bits T2I0, T2I1, and T2PS as listed in **figure 7-35**.

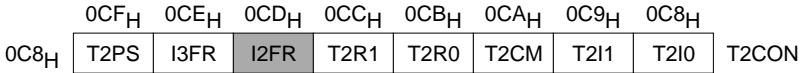
Reload of Timer 2 (see figure 7-33 b)


The reload mode for timer 2 is selected by bits T2R0 and T2R1 in SFR T2CON as listed in **figure 7-35**. Two reload modes are selectable:

In mode 0, when timer 2 rolls over from all 1's to all 0's, it not only sets TF2 but also causes the timer 2 registers to be loaded with the 16-bit value in the CRC register, which is preset by software. The reload will happen in the same machine cycle in which TF2 is set, thus overwriting the count value 0000_H.

In mode 1, a 16-bit reload from the CRC register is caused by a negative transition at the corresponding input pin T2EX/P1.5. In addition, this transition will set flag EXF2, if bit EXEN2 in SFR IEN1 is set. If the timer 2 interrupt is enabled, setting EXF2 will generate an interrupt (more about interrupts in section 8). The external input pin T2EX is sampled in every machine cycle. When the sampling shows a high in one cycle and a low in the next cycle, a transition will be recognized. The reload of timer 2 registers will then take place in the cycle following the one in which the transition was detected.

Figure 7-34
Special Function Register T2CON



 These bits are not used in combination with timer 2.

Timer 2 control register. Bit-addressable register which controls timer 2 function and compare mode of registers CRC, CC1 to CC3.

Bit	Symbol	Symbol
T2I1	T2I0	Timer 2 input selection
0	0	No input selected, timer 2 stops
0	1	Timer function
1	0	input frequency = $f_{osc}/12$ (T2PS = 0) or $f_{osc}/24$ (T2PS = 1)
1	1	Counter function, external input signal at pin T2/P1.7
		Gated timer function, input controlled by pin T2/P1.7
T2R1	T2R0	Timer 2 reload mode selection
0	X	Reload disabled
1	0	Mode 0: auto-reload upon timer 2 overflow (TF2)
1	1	Mode 1: reload upon falling edge at pin T2EX/P1.5
	T2CM	Compare mode bit for registers CRC, CC1 through CC3. When set, compare mode 1 is selected. T2CM = 0 selects compare mode 0.
	I3FR	External interrupt 3 falling/rising edge flag, also used for capture function in combination with register CRC (see section 7.5.3). If set, capture to register to CRC (if enabled) will occur on a positive transition at pin P1.0/INT3/CC0. If I3FR is cleared, capture will occur on a negative transition.
	T2PS	Prescaler select bit. When set, timer 2 is clocked in the "timer" or "gated timer" function with 1/24 of the oscillator frequency. T2PS = 0 gates $f_{osc}/12$ to timer 2. T2PS must be 0 for the counter operation of timer 2.

7.5.2 Compare Function of Registers CRC, CC1 to CC3

The compare function of a timer/register combination can be described as follows. The 16-bit value stored in a compare/capture register is compared with the contents of the timer register. If the count value in the timer register matches the stored value, an appropriate output signal is generated at a corresponding port pin, and an interrupt is requested.

The contents of a compare register can be regarded as 'time stamp' at which a dedicated output reacts in a predefined way (either with a positive or negative transition). Variation of this 'time stamp' somehow changes the wave of a rectangular output signal at a port pin. This may - as a variation of the duty cycle of a periodic signal - be used for pulse width modulation as well as for a continually controlled generation of any kind of square wave forms. In the case of the SAB 80(C)515, two compare modes are implemented to cover a wide range of possible applications.

The compare modes 0 and 1 are selected by bit T2CM in special function register T2CON (see **figure 7-34**). In both compare modes, the new value arrives at the port pin 1 within the same machine cycle in which the internal compare signal is activated.

The four registers CRC, CC1 to CC3 are multifunctional as they additionally provide a capture, compare or reload capability (the CRC register only, see section 7.5.1). A general selection of the function is done in register CCEN (see **figure 7-40**). Please note that the compare interrupt CC0 can be programmed to be negative or positive transition activated. The internal compare signal (not the output signal at the port pin!) is active as long as the timer 2 contents is equal to the one of the appropriate compare registers, and it has a rising and a falling edge. Thus, when using the CRC register, it can be selected whether an interrupt should be caused when the compare signal goes active or inactive, depending on bit I3FR in T2CON. For the CC registers 1 to 3 an interrupt is always requested when the compare signal goes active (see **figure 7-36**).

7.5.2.1 Compare Mode 0

In mode 0, upon matching the timer and compare register contents, the output signal changes from low to high. It goes back to a low level on timer overflow. As long as compare mode 0 is enabled, the appropriate output pin is controlled by the timer circuit only, and not by the user. Writing to the port will have no effect. **Figure 7-35** shows a functional diagram of a port latch in compare mode 0. The port latch is directly controlled by the two signals timer overflow and compare. The input line from the internal bus and the write-to-latch line are disconnected when compare mode 0 is enabled.

Compare mode 0 is ideal for generating pulse width modulated output signals, which in turn can be used for digital-to-analog conversion via a filter network or by the controlled device itself (e.g. the inductance of a DC or AC motor). Mode 0 may also be used for providing output clocks with initially defined period and duty cycle. This is the mode which needs the least CPU time. Once set up, the output goes on oscillating without any CPU intervention. **Figure 7-36** and **7-37** illustrate the function of compare mode 0.

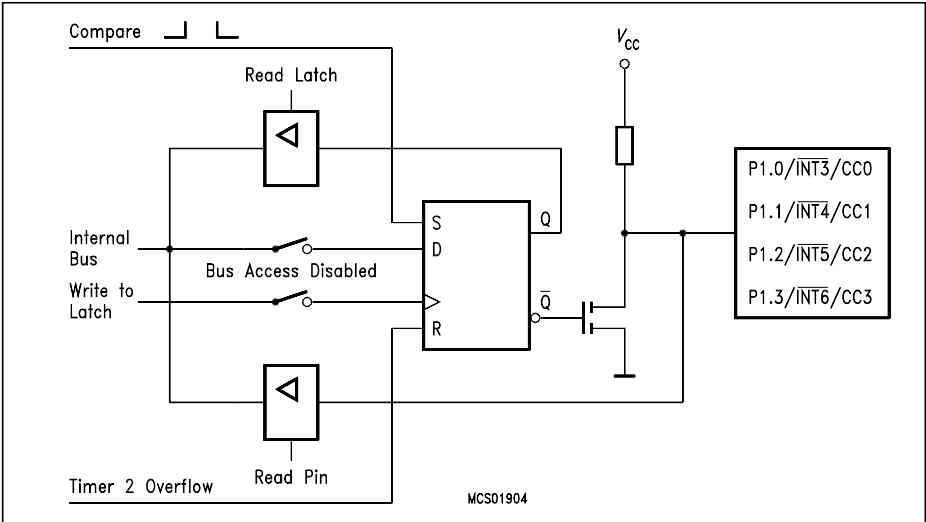


Figure 7-35
Port Latch in Compare Mode 0

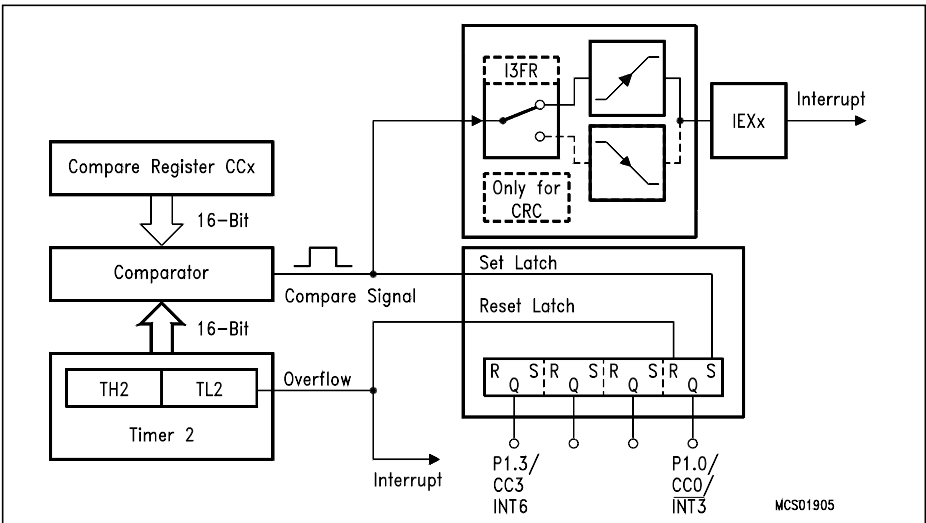


Figure 7-36
Timer 2 with Registers CCx in Compare Mode 0
(CCx stands for CRC, CC1 to CC3; IEXx stands for IEX3 to IEX6)

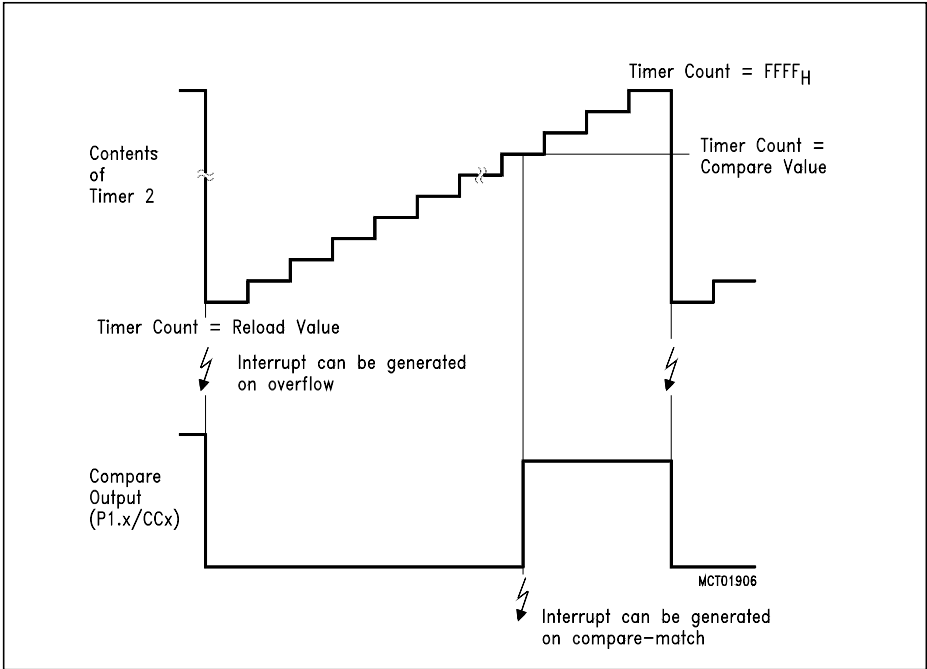


Figure 7-37
Function of Compare Mode 0

Modulation Range in Compare Mode 0

Generally it can be said that for every PWM generation in compare mode 0 with n-bit wide compare registers there are 2ⁿ different settings for the duty cycle. Starting with a constant low level (0% duty cycle) as the first setting, the maximum possible duty cycle then would be

$$(1 - 1/2^n) \times 100\%$$

This means that a variation of the duty cycle from 0% to real 100% can never be reached if the compare register and timer register have the same length. There is always a spike which is as long as the timer clock period.

This "spike" may either appear when the compare register is set to the reload value (limiting the lower end of the modulation range) or it may occur at the end of a timer period. In a timer 2/CCx register configuration in compare mode 0 this spike is divided into two halves: one at the beginning when the contents of the compare register is equal to the reload value of the timer; the other half when the compare register is equal to the maximum value of the timer register (here: 0FFF_H). Please refer to **figure 7-38** where the maximum and minimum duty cycle of a compare output signal is illustrated. Timer 2 is incremented with the machine clock ($f_{osc}/12$), thus at 12-MHz operational frequency, these spikes are both approx. 500 ns long.

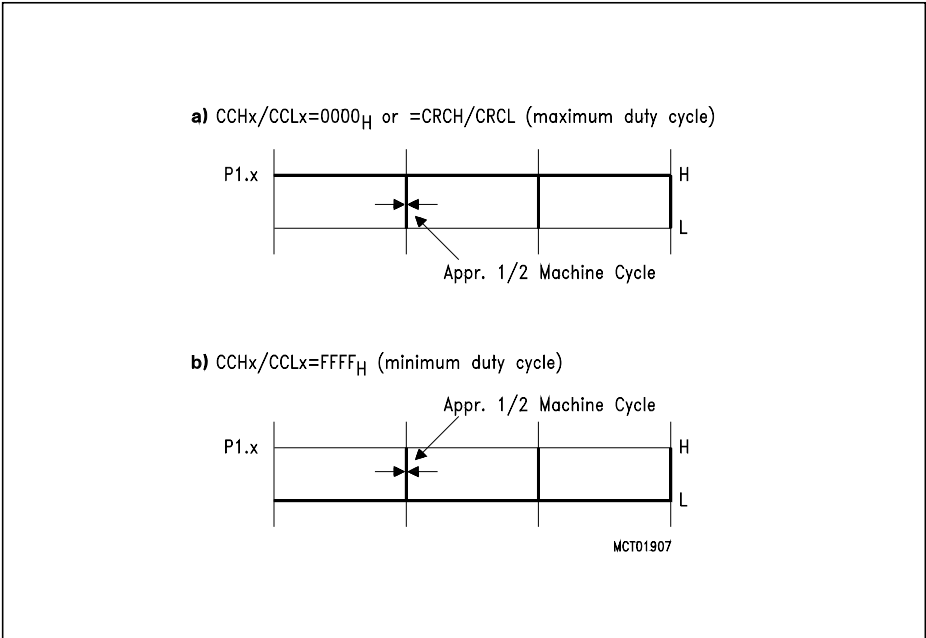


Figure 7-38
Modulation Range of a PWM Signal, Generated with a
Timer 2/CCx Register Combination in Compare Mode 0

The following example shows how to calculate the modulation range for a PWM signal. To calculate with reasonable numbers, a reduction of the resolution to 8-bit is used. Otherwise (for the maximum resolution of 16-bit) the modulation range would be so severely limited that it would be negligible.

Example:

Timer 2 in auto-reload mode; contents of reload register CRC = 0FF00_H

$$\text{Restriction of modulation range} = \frac{1}{256 \times 2} \times 100\% = 0.195\%$$

This leads to a variation of the duty cycle from 0.195% to 99.805% for a timer 2/CCx register configuration when 8 of 16 bits are used.

7.5.2.2 Compare Mode 1

In compare mode 1, the software adaptively determines the transition of the output signal. It is commonly used when output signals are not related to a constant signal period (as in a standard PWM generation) but must be controlled very precisely with high resolution and without jitter. In compare mode 1, both transitions of a signal can be controlled. Compare outputs in this mode can be regarded as high speed outputs which are independent of the CPU activity.

If mode 1 is enabled, and the software writes to the appropriate output latch at the port, the new value will not appear at the output pin until the next compare match occurs. Thus, one can choose whether the output signal is to make a new transition (1-to-0 or 0-to-1, depending on the actual pin-level) or should keep its old value at the time the timer 2 count matches the stored compare value.

Figure 7-39 shows a functional diagram of a timer/compare register/port latch configuration in compare mode 1. In this function, the port latch consists of two separate latches. The upper latch (which acts as a "shadow latch") can be written under software control, but its value will only be transferred to the output latch (and thus to the port pin) in response to a compare match.

Note that the double latch structure is transparent as long as the internal compare signal is active. While the compare signal is active, a write operation to the port will then change both latches. This may become important when driving timer 2 with a slow external clock. In this case the compare signal could be active for many machine cycles in which the CPU could unintentionally change the contents of the port latch. For details see also section 7.5.2.3 "Using Interrupts in Combination with the Compare Function".

A read-modify-write instruction (see section 7.1) will read the user-controlled "shadow latch" and write the modified value back to this "shadow-latch". A standard read instruction will - as usual - read the pin of the corresponding compare output.

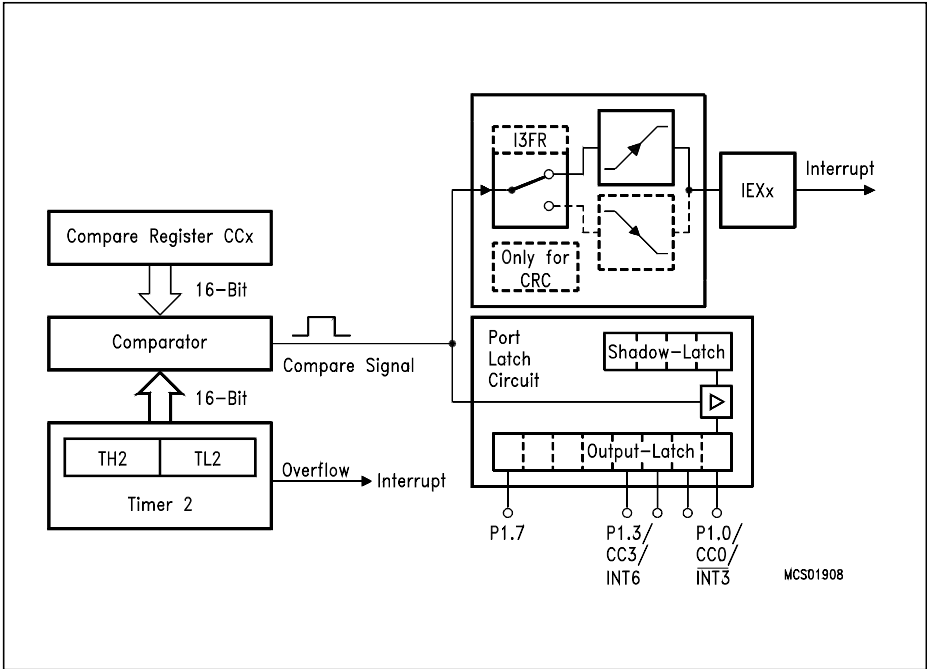
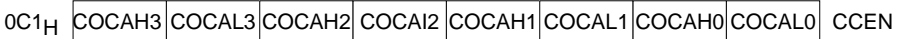


Figure 7-39
Timer 2 with Registers CCx in Compare Mode 1
 (CCx stands for CRC, CC1 to CC3; IEXx stands for IEX3 to IEX6)

Figure 7-40
Special Function Register CCEN



Compare/capture enable register selects compare or capture function for register CRC, CC1 to CC3.

Bit		Function
COCAH0	COCAL0	Compare/capture mode for CRC register
0	0	Compare/capture disabled
0	1	Capture on falling/rising edge at pin P1.0/INT3/CC0
1	0	Compare enabled
1	1	Capture on write operation into register CRCL
COCAH1	COCAL1	Compare/capture mode for CC register 1
0	0	Compare/capture disabled
0	1	Capture on rising edge at pin P1.1/INT4/CC1
1	0	Compare enabled
1	1	Capture on write operation into register CCL1
COCAH2	COCAL2	Compare/capture mode for CC register 2
0	0	Compare/capture disabled
0	1	Capture on rising edge at pin P1.2/INT5/CC2
1	0	Compare enabled
1	1	Capture on write operation into register CCL2
COCAH3	COCAL3	Compare/capture mode for CC register 3
0	0	Compare/capture disabled
0	1	Capture on rising edge at pin P1.3/INT6/CC3
1	0	Compare enabled
1	1	Capture on write operation into register CCL3

7.5.2.3 Using Interrupts in Combination with the Compare Function

The compare service of registers CRC, CC1, CC2 and CC3 is assigned to alternate output functions at port pins P1.0 to P1.3. Another option of these pins is that they can be used as external interrupt inputs. However, when using the port lines as compare outputs then the input line from the port pin to the interrupt system is disconnected (but the pin’s level can still be read under software control). Thus, a change of the pin’s level will not cause a setting of the corresponding interrupt flag. In this case, the interrupt input is directly connected to the (internal) compare signal thus providing a **compare interrupt**.

The compare interrupt can be used very effectively to change the contents of the compare registers or to determine the level of the port outputs for the next "compare match". The principle is, that the internal compare signal (generated at a match between timer count and register contents) not only manipulates the compare output but also sets the corresponding interrupt request flag. Thus, the current task of the CPU is interrupted - of course provided the priority of the compare interrupt is higher than the present task priority - and the corresponding interrupt service routine is called. This service routine then sets up all the necessary parameters for the next compare event.

Some **advantages** in using compare interrupts:

Firstly, there is no danger of unintentional overwriting a compare register before a match has been reached. This could happen when the CPU writes to the compare register without knowing about the actual timer 2 count.

Secondly, and this is the most interesting advantage of the compare feature, the output pin is exclusively controlled by hardware therefore completely independent from any service delay which in real time applications could be disastrous. The compare interrupt in turn is not sensitive to such delays since it loads the parameters for the next event. This in turn is supposed to happen after a sufficient space of time.

Please note two special cases where a program using compare interrupts could show a "surprising" behavior:

The first configuration has already been mentioned in the description of compare mode 1. The fact that the compare interrupts are transition activated becomes important when driving timer 2 with a slow external clock. In this case it should be carefully considered that the compare signal is active as long as the timer 2 count is equal to the contents of the corresponding compare register, and that the compare signal has a rising and a falling edge. Furthermore, the "shadow latches" used in compare mode 1 are transparent while the compare signal is active.

Thus, with a slow input clock for timer 2, the comparator signal is active for a long time (= high number of machine cycles) and therefore a fast interrupt controlled reload of the compare register could not only change the "shadow latch" - as probably intended - but also the output buffer.

When using the CRC , you can select whether an interrupt should be generated when the compare signal goes active or inactive, depending on the status of bit I3FR in T2CON (see figure 8-5).

Initializing the interrupt to be negative transition triggered is advisable in the above case. Then the compare signal is already inactive and any write access to the port latch just changes the contents of the "shadow-latch".

Please note that for CC registers 1 to 3 an interrupt is always requested when the compare signal goes active.

The second configuration which should be noted is when compare function is combined with negative transition activated interrupts. If the port latch of port P1.0 contains a 1, the interrupt request flags IEX2 will immediately be set after enabling the compare mode for the CRC register. The reason is that first the external interrupt input is controlled by the pin's level. When the compare option is enabled the interrupt logic input is switched to the internal compare signal, which carries a low level when no true comparison is detected. So the interrupt logic sees a 1-to-0 edge and sets the interrupt request flag.

An unintentional generation of an interrupt during compare initialization can be prevented if the request flag is cleared by software after the compare is activated and before the external interrupt is enabled.

7.5.3 Capture Function

Each of the three compare/capture registers CC1 to CC3 and the CRC register can be used to latch the current 16-bit value of the timer 2 registers TL2 and TH2. Two different modes are provided for this function. In mode 0, an external event latches the timer 2 contents to a dedicated capture register. In mode 1, a capture will occur upon writing to the low order byte of the dedicated 16-bit capture register. This mode is provided to allow the software to read the timer 2 contents "on-the-fly".

In mode 0, the external event causing a capture is

- for CC registers 1 to 3: a positive transition at pins CC1 to CC3 of port 1
- for the CRC register: a positive or negative transition at the corresponding pin, depending on the status of the bit I3FR in SFR T2CON. If the edge flag is cleared, a capture occurs in response to a negative transition; if the edge flag is set a capture occurs in response to a positive transition at pin P1.0/INT3/ CC0.

In both cases the appropriate port 1 pin is used as input and the port latch must be programmed to contain a one (1). The external input is sampled in every machine cycle. When the sampled input shows a low (high) level in one cycle and a high (low) in the next cycle, a transition is recognized. The timer 2 contents is latched to the appropriate capture register in the cycle following the one in which the transition was identified.

In mode 0 a transition at the external capture inputs of registers CC0 to CC3 will also set the corresponding external interrupt request flags IEX3 to IEX6. If the interrupts are enabled, an external capture signal will cause the CPU to vector to the appropriate interrupt service routine.

In mode 1 a capture occurs in response to a write instruction to the low order byte of a capture register. The write-to-register signal (e.g. write-to-CRCL) is used to initiate a capture. The value written to the dedicated capture register is irrelevant for this function. The timer 2 contents will be latched into the appropriate capture register in the cycle following the write instruction. In this mode no interrupt request will be generated.

Figures 7-41 and 7-42 show functional diagrams of the capture function of timer 2. Figure 7-41 illustrates the operation of the CRC register, while figure 7-42 shows the operation of the compare/capture registers 1 to 3.

The two capture modes can be established individually for each capture register by bits in SFR CCEN (compare/capture enable register). That means, in contrast to the compare modes, it is possible to simultaneously select mode 0 for one capture register and mode 1 for another register. The bit positions and functions of CCEN are listed in figure 7-40.

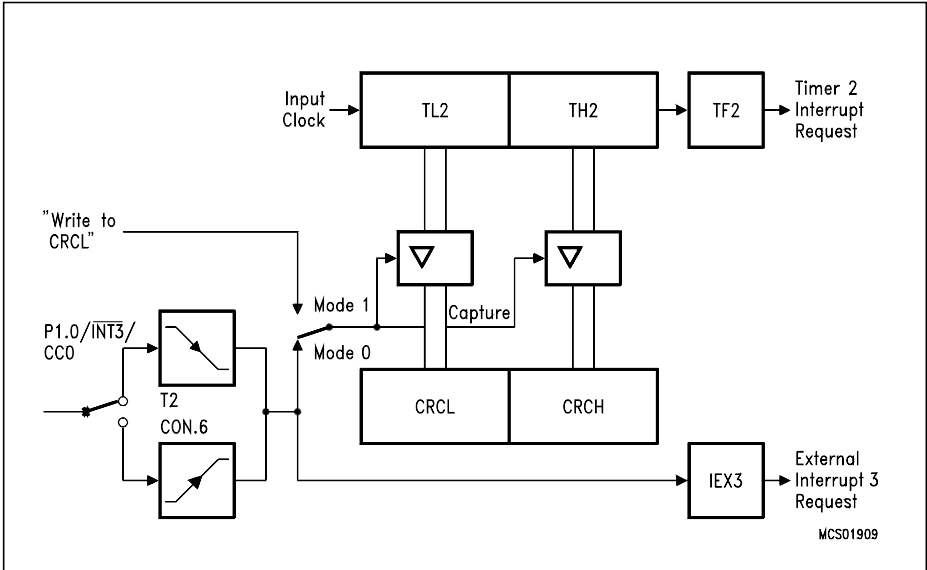


Figure 7-41
Capture with Register CRC

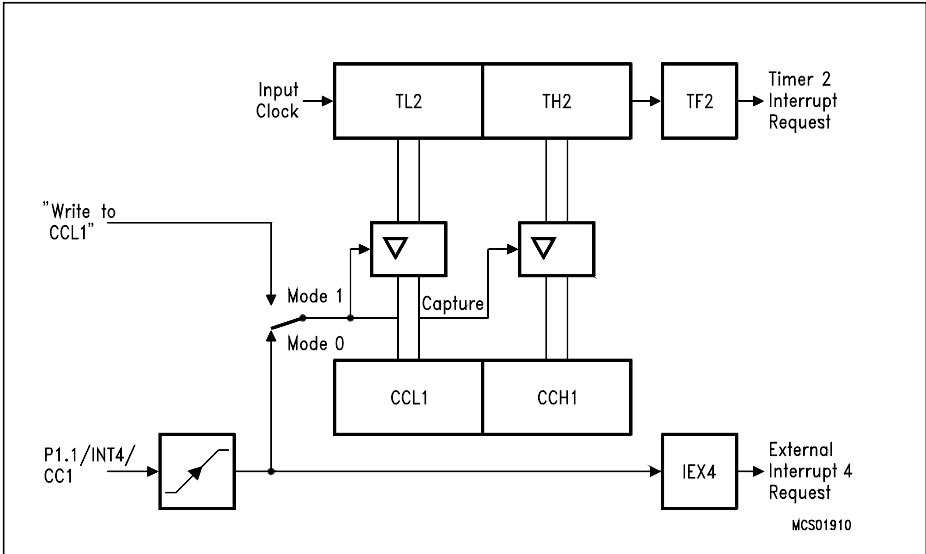


Figure 7-42
Capture with Registers CC1 to CC3

7.6 Power Saving Modes

For significantly reducing power consumption, the SAB 80(C)515/80(C)535 provides two Power Saving Modes:

- The Power-Down Mode
 Operation of the component stops completely, the oscillator is turned off. Only the internal RAM is supplied with a very low standby current.
- The Idle Mode (SAB 80C515/80C535 only)
 The CPU is gated off from the oscillator. All peripherals are further supplied by the oscillator clock and are able to do their jobs.

These modes are described separately for each component in the following sections.

There are numerous applications which require high system security and at the same time reliability in electrically noisy environments. In such applications unintentional entering of the power saving modes must be absolutely avoided. A power saving mode would reduce the controller's performance (in case of idle mode) or even stop each operation (in case of power-down mode). This situation might be fatal for the system. Such critical applications often use the watchdog timer to prevent the system from program upsets. Then accidental entering of the power saving modes would even stop the watchdog timer and would circumvent the task of system reliability.

The SAB 80C515/80C535 provides software and hardware protection against accidental entering as described above (see chapter 7.6.2).

7.6.1 Power Saving Modes of the SAB 80515/80535

The SAB 80515/80535 allows a reduction of the power consumption using the power-down mode.

7.6.1.1 Power-Down Mode of the SAB 80515/80535

The power-down mode in the SAB 80515/80535 allows a reduction of V_{CC} , to zero while saving 40 bytes of the on-chip RAM through a backup supply connected to the V_{PD} pin. In the following, the terms V_{CC} and V_{PD} are used to specify the voltages on pin V_{CC} and pin V_{PD} , respectively.

If $V_{CC} > V_{PD}$, the 40 bytes are supplied from V_{CC} . V_{PD} may then below. If $V_{CC} < V_{PD}$, the current for the 40 bytes is drawn from V_{PD} . The addresses of these backup-powered RAM locations range from 88 to 127 (58_H to 7F_H). The current drawn from the backup power supply is typically 1 mA, max. 3 mA.

To utilize this feature, the user's system - upon detecting an imminent power failure - would interrupt the processor in some manner to transfer relevant data to the 40-byte on-chip RAM and enable the backup power supply to the V_{PD} pin. Then a reset should be accomplished before V_{CC} falls below its operating limit. When power returns, a power-on reset should be accomplished, and the backup supply needs to stay on long enough to resume normal operation. **Figure 7-43** illustrates the timing upon a power failure.

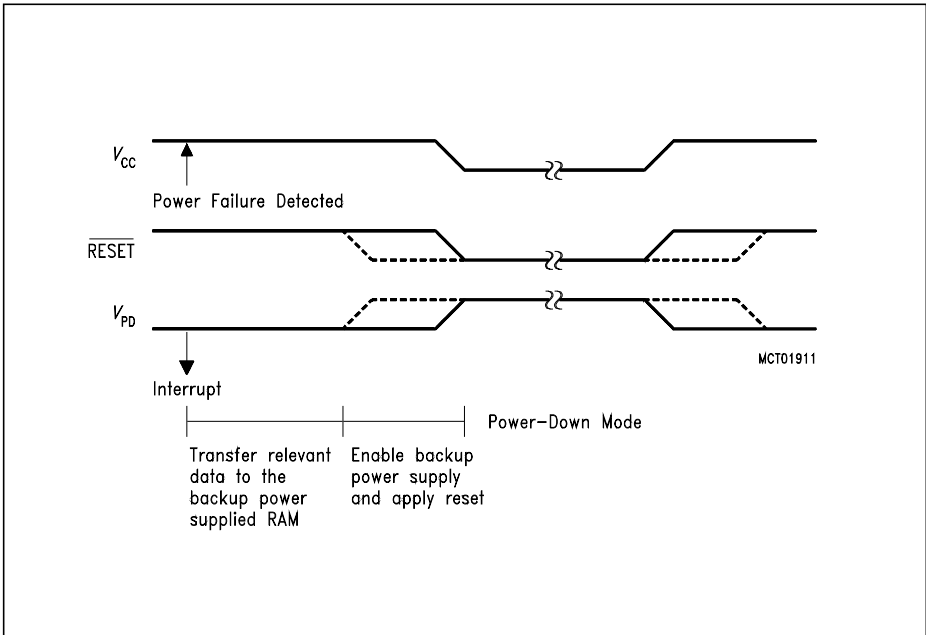


Figure 7-43 Reset and RAM Backup Power Timing of the SAB 80515/80535

7.6.2 Power Saving Modes of the SAB 80515/80535

Differences between the Power-Down Modes of the SAB 80C515/80C535 and the SAB 80515/80535

The power-down mode of the SAB 80515/80535 allows retention of 40 bytes on-chip RAM through a backup supply connected to the V_{PD} pin.

The ACMOS versions SAB 80C515/80C535 have the following additional features:

- The same power supply (pin V_{CC}) for active, power-down (retention of the whole int. RAM area), and idle mode.
- An extra pin (\overline{PE}) that allows enabling/disabling of the power-saving modes.
- A software protection that enables the power saving modes via special function register PCON (Power Control Register).

Hardware Enable for the Use of the Power Saving Modes

To provide power saving modes together with effective protection against unintentional entering of these modes, the SAB 80C515/80C535 has an extra pin for disabling the use of the power saving modes. This pin is called \overline{PE} (power saving enable), and its function is as follows:

$\overline{PE} = 1$ (logic high level): Use of the power saving modes is not possible. The instruction sequences used for entering these modes will not affect the normal operation of the device.

$\overline{PE} = 0$ (logic low level): All power saving modes can be activated as described in the following sections.

When left unconnected, the pin \overline{PE} is pulled to high level by a weak internal pullup. This is done to provide system protection by default.

In addition to the hardware enable/disable of the power saving modes, a double-instruction sequence which is described in the corresponding sections is necessary to enter power-down and idle mode. The combination of all these safety precautions provides a maximum of system protection.

Application Example for Switching Pin \overline{PE}

For most applications in noisy environments, certain components external to the chip are used to give warning of a power failure or a turn off of the power supply.

These circuits could be used to control the \overline{PE} pin. The possible steps to go into power-down mode could then be as follows:

- A power-fail signal forces the controller to go into a high priority interrupt routine. This interrupt routine saves the actual program status. At the same time pin \overline{PE} is pulled low by the power-fail signal.
- Finally the controller enters power-down mode by executing the relevant double-instruction sequence.

7.6.2.1 Power-Down Mode of the SAB 80C515/80C535

In the power-down mode, the on-chip oscillator is stopped. Therefore, all functions are stopped, only the contents of the on-chip RAM and the SFR's are held. The port pins controlled by their port latches output the values that are held by their SFR'S. The port pins which serve the alternate output functions show the values they had at the end of the last cycle of the instruction which initiated the power-down mode; when enabled, the clockout signal (P1.6/CLKOUT) will stop at low level. ALE and $\overline{\text{PSEN}}$ are held at logic low level (see table 7-10).

If the power-down mode is to be used, the pin $\overline{\text{PE}}$ must be held low. Entering the power-down mode is done by two consecutive instructions immediately following each other. The first instruction has to set the flag bit PDE (PCON.1) and must not set bit PDS (PCON.6). The following instruction has to set the start bit PDS (PCON.6) and must not set bit PDE (PCON.1). The hardware ensures that a concurrent setting of both bits, PDE and PDS, will not initiate the power-down mode. Bit PDE and PDS will automatically be cleared after having been set and the value shown when reading one of these bits is always zero (0). **Figure 7-44** shows the special function register PCON. This double-instruction sequence is implemented to minimize the chance of unintentionally entering the power-down mode, which could possibly "freeze" the chip's activity in an undesired status.

Note that PCON is not a bit-addressable register, so the above mentioned sequence for entering the power-down mode is composed of byte handling instructions.

The following instruction sequence may serve as an example:

```
ORL   PCON,#00000010B   ;Set bit PDE,
                          ;bit PDS must not be set
ORL   PCON,#01000000B   ;Set bit PDS,
                          ;bit PDE must not be set
```

The instruction that sets bit PDS is the last instruction executed before going into power-down mode. If idle mode and power-down mode are invoked simultaneously, the power-down mode takes precedence.

The only exit from power-down mode is a hardware reset. Reset will redefine all SFR'S, but will not change the contents of the internal RAM.

In the power-down mode, V_{CC} can be reduced to minimize power consumption. Care must be taken, however, to ensure that V_{CC} is not reduced before the power-down mode is invoked, and that V_{CC} is restored to its normal operating level before the power-down mode is terminated. The reset signal that terminates the power-down mode also frees the oscillator. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize (similar to power-on reset).

7.6.2.2 Idle Mode of the SAB 80C515/80C535

In idle mode the oscillator of the SAB 80C515/80C535 continues to run, but the CPU is gated off from the clock signal. However, the interrupt system, the serial channel, the A/D converter and all timers, except for the watchdog timer, are further provided with the clock. 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 mode.

The reduction of power consumption, which can be achieved by this feature, depends on the number of peripherals running. If all timers are stopped and the A/D converter and the serial interface are not running, maximum power reduction can be achieved. This state is also the test condition for the idle I_{CC} (see the DC characteristics in the data sheet).

Thus, the user has to make sure that the right peripheral continues to run or is stopped, respectively, during idle. Also, the state of all port pins - either the pins controlled by their latches or controlled by their secondary functions - depends on the status of the controller when entering idle.

Normally the port pins hold the logical state they had at the time idle was activated. If some pins are programmed to serve their alternate functions they still continue to output during idle if the assigned function is on. This applies for the compare outputs as well as for the system clock output signal and the serial interface in case the latter could not finish reception or transmission during normal operation. The control signals ALE and \overline{PSEN} are held at logic high levels (see table 7-10).

During idle, as in normal operating mode, the ports can be used as inputs. Thus, a capture or reload operation as well as an A/D conversion can be triggered, the timers can be used to count external events and external interrupts can be detected.

Table 7-10
Status of External Pins During Idle and Power-Down Mode

Outputs	Last Instruction Executed from Internal Code Memory		Last Instruction Executed from External Code Memory	
	Idle	Power-down	Idle	Power-down
ALE	High	Low	High	Low
PSEN	High	Low	High	Low
Port 0	Data	Data	Float	Float
Port 1	Data/alternate outputs	Data/last output	Data/alternate outputs	Data/last output
Port 2	Data	Data	Address	Data
Port 3	Data/alternate outputs	Data/last output	Data/alternate outputs	Data/last output
Port 4	Data	Data	Data	Data
Port 5	Data	Data	Data	Data

The watchdog timer is the only peripheral which is automatically stopped during idle. The idle mode makes it possible to "freeze" the processor's status for a certain time or until an external event causes the controller to go back into normal operating mode. Since the watchdog timer is stopped during idle mode, this useful feature of the SAB 80C515/80C535 is provided even if the watchdog function is used simultaneously.

If the idle mode is to be used the pin \overline{PE} must be held low. Entering the idle mode is to be done by two consecutive instructions immediately following each other. The first instruction has to set the flag bit IDLE (PCON.0) and must not set bit IDLS (PCON.5), the following instruction has to set the start bit IDLS (PCON.5) and must not set bit IDLE (PCON.0). The hardware ensures that a concurrent setting of both bits, IDLE and IDLS will not initiate the idle mode. Bits IDLE and IDLS will automatically be cleared after having been set. If one of these register bits is read the value shown is zero (0). **Figure 7-44** shows special function register PCON. This double-instruction sequence is implemented to minimize the chance of unintentionally entering the idle mode.

Note that PCON is not a bit-addressable register, so the above mentioned sequence for entering the idle mode is to be done by byte handling instructions.

The following instruction sequence may serve as an exemple:

```

ORL   PCON,#00000001B   ;Set bit IDLE,
                               ;bit IDLS must not be set

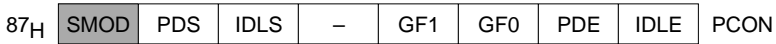
ORL   PCON,#00100000B   ;Set bit IDLS,
                               ;bit IDLE must not be set
    
```


The instruction that sets bit IDLS is the last instruction executed before going into idle mode.

Terminating the Idle Mode

- The idle mode can be terminated by activation of any enabled interrupt. The CPU operation is resumed, the interrupt will be serviced and the next instruction to be executed after the RETI instruction will be the one following the instruction that set the bit IDLS.
- The other possibility of terminating the idle mode is a hardware reset. Since the oscillator is still running, the hardware reset is held active for only two machine cycles for a complete reset.

Figure 7-44
Special Function Register PCON (Address 87_H) of the SAB 80C515/80C535



 These bits are not used in controlling the power saving modes.

Bit	Function
PDS	Power-down start bit. The instruction that sets the PDS flag bit is the last instruction before entering the power-down mode.
IDLS	IDLE start bit. The instruction that sets the IDSL flag bit is the last instruction before entering the idle mode.
GF1	General purpose flag
GF0	General purpose flag
PDE	Power-down enable bit. When set, starting the power-down mode is enabled.
IDLE	Idle mode enable bit. When set, starting the idle mode is enabled.

7.7 Watchdog Timer

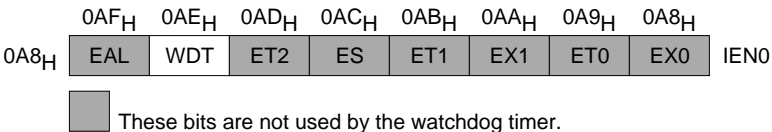
As a means of graceful recovery from software or hardware upset a watchdog timer is provided in the SAB 80(C)515/80(C)535. If the software fails to clear the watchdog timer at least every 65532 μ s, an internal hardware reset will be initiated. The software can be designed such that the watchdog times out if the program does not progress properly. The watchdog will also time out if the software error was due to hardware-related problems. This prevents the controller from malfunctioning for longer than 65 ms if a 12-MHz oscillator is used.

The watchdog timer is a 16-bit counter which is incremented once every machine cycle. After an external reset the watchdog timer is disabled and cleared to 0000_H. The counter is started by setting bit SWDT (bit 6 in SFR IEN1). After having been started, the bit WDTS (watchdog timer status, bit 6 in SFR IPO) is set. Note that the watchdog timer cannot be stopped by software. It can only be cleared to 0000_H by first setting bit WDT (IEN0.6) and with the next instruction setting SWDT. Bit WDT will automatically be cleared during the second machine cycle after having been set. For this reason, setting SWDT bit has to be a one cycle instruction (e.g. SETB SWDT). This double instruction clearing of the watchdog timer was implemented to minimize the chance of unintentionally clearing the watchdog. To prevent the watchdog from overflowing, it must be cleared periodically.

Starting the watchdog timer by setting only bit SWDT does not reload the WDTREL register to the watchdog timer registers WDTL/WDTH. A reload occurs only by using the double instruction refresh sequence SETB WDT / SETB SWDT.

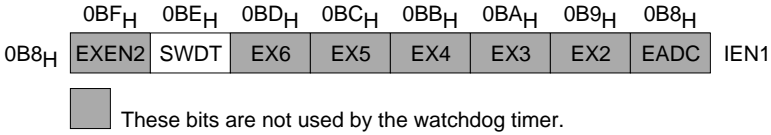
If the software fails to clear the watchdog in time, an internally generated watchdog reset is entered at the counter state FFFC_H, which lasts four machine cycles. This internal reset differs from an external reset only to the extent that the watchdog timer is not disabled. Bit WDTS (was set by starting WDT) allows the software to examine from which source the reset was initiated. If it is set, the reset was caused by a watchdog timer overflow.

Figure 7-45
Special Function Register IEN0



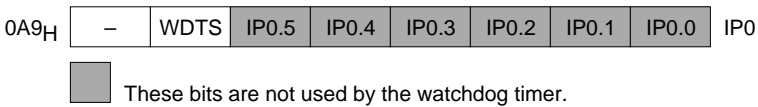
Bit	Function
WDT	Watchdog timer refresh flag. Set to initiate a refresh of the watchdog timer. Must be set directly before SWDT is set to prevent an unintentional refresh of the watchdog timer. WDT is reset by hardware two processor cycles after it has been set.

Figure 7-46
Special Function Register IEN1



Bit	Function
SWDT	Watchdog timer start/refresh flag. Set to activate/refresh the watchdog timer. When directly set after setting WDT, a watchdog timer refresh is performed. Bit SWDT is reset by hardware two processor cycles after it has been set.

Figure 7-47
Special Function Register IP0



Bit	Function
WDTS	Watchdog timer status flag. Set by hardware when the watchdog timer was started. Can be read by software.

7.8 Oscillator and Clock Circuit

XTAL1 and XTAL2 are the input and output of a single-stage on-chip inverter which can be configured with off-chip components as a Pierce oscillator. The oscillator, in any case, drives the internal clock generator. The clock generator provides the internal clock signals to the chip at half the oscillator frequency. These signals define the internal phases, states and machine cycles, as described in chapter 3.

7.8.1 Crystal Oscillator Mode

Figure 7-48 shows the recommended oscillator circuit.

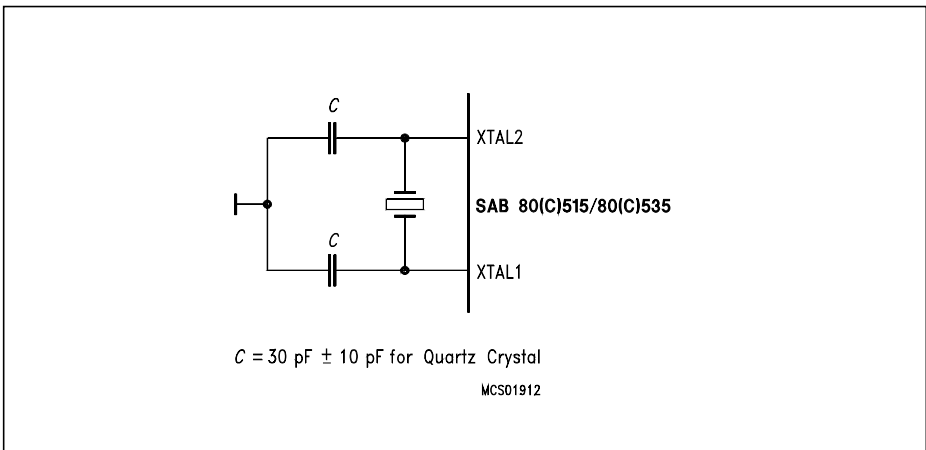


Figure 7-48
Recommended Oscillator Circuit for the SAB 80(C)515/80(C)535

In this application the on-chip oscillator is used as a crystal-controlled, positive-reactance oscillator (a more detailed schematic is given in figure 7-49 and 7-51). It is operated in its fundamental response mode as an inductive reactor in parallel resonance with a capacitor external to the chip. The crystal specifications and capacitances are non-critical. In this circuit 30 pF can be used as single capacitance at any frequency together with a good quality crystal. A ceramic resonator can be used in place of the crystal in cost-critical applications. If a ceramic resonator is used, C_1 and C_2 are normally selected to be different values. We recommend consulting the manufacturer of the ceramic resonator for value specifications of these capacitors.

7.8.2 Driving for External Source

The SAB 80(C)515/80(C)535 can be driven from an external oscillator.

Please note that there is a difference between driving MYMOS and ACMOS devices from an external clock source.

7.8.2.1 Driving the SAB 80515/80535 from External Source

For driving the SAB 80515/80535 from an external clock source, the external clock signal is to be applied to XTAL2. A pullup resistor is recommended to increase the noise margin, but is optional if the output high level of the driving gate meets the V_{IH1} specification of XTAL2.

XTAL1 has to be connected to ground (see figure 7-50).

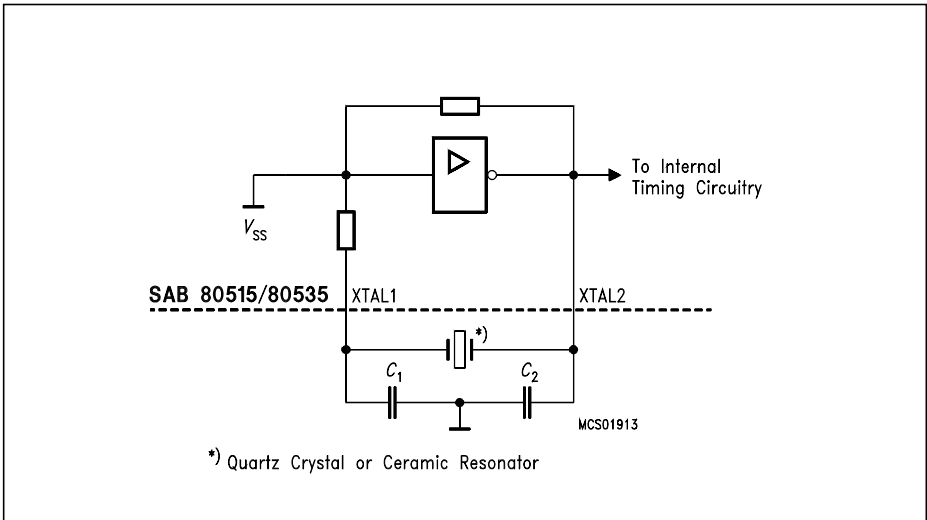


Figure 7-49
On-Chip Oscillator Circuitry

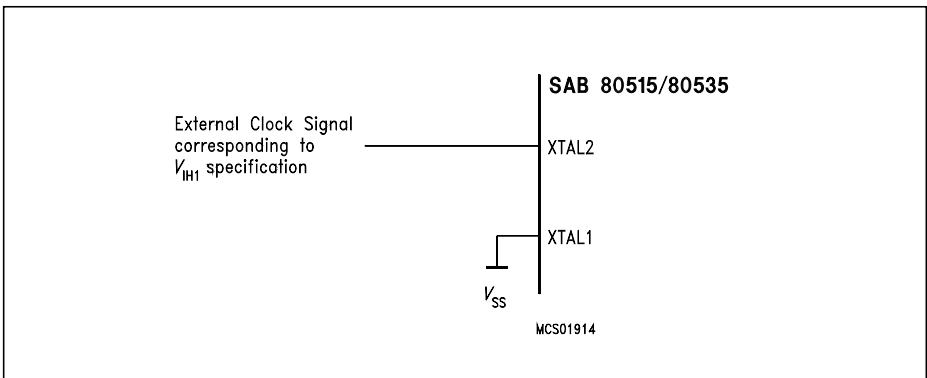


Figure 7-50
External Clock Source

7.8.2.2 Driving the SAB 80C515/80C535 from External Source

For driving the SAB 80C515/80C535 from an external clock source, the external clock signal is to be applied to XTAL2, as shown in **figure 7-52**. A pullup resistor is recommended, but is optional if the output high level of the driving gate corresponds to the V_{IH1} specification of XTAL2.

XTAL1 has to be left unconnected.

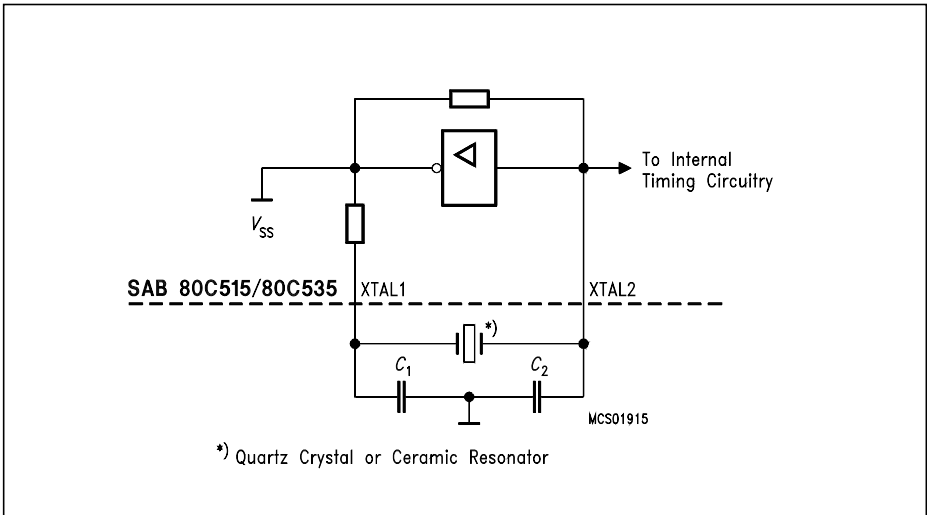


Figure 7-51
On-Chip Oscillator Circuitry

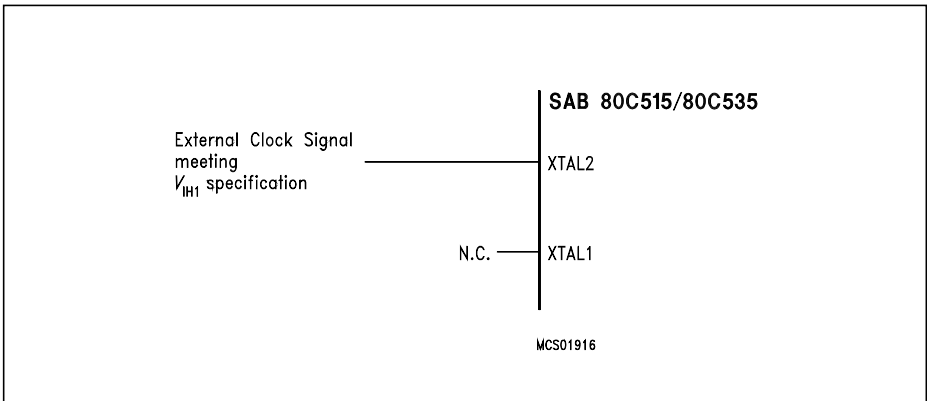
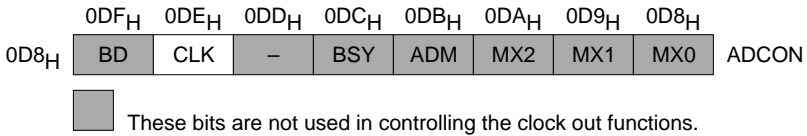


Figure 7-52
External Clock Source

7.9 System Clock Output

For peripheral devices requiring a system clock, the SAB 80(C)515/80(C)535 provides a clock output signal derived from the oscillator frequency as an alternate output function on pin P1.6/CLKOUT. If bit CLK is set (bit 6 of special function register ADCON, **see figure 7-53**), a clock signal with 1/12 of the oscillator frequency is gated to pin P1.6/CLKOUT. To use this function the port pin must be programmed to a one (1), which is also the default after reset.

Figure 7-53
Special Function Register ADCON (Address 0D8H)



Bit	Function
CLK	Clockout enable bit. When set, pin P1.6/CLKOUT outputs the system clock which is 1/12 of the oscillator frequency.

The system clock is high during S3P1 and S3P2 of every machine cycle and low during all other states. Thus, the duty cycle of the clock signal is 1:6. Associated with a MOVX instruction the system clock coincides with the last state (S3) in which a \overline{RD} or \overline{WR} signal is active. A timing diagram of the system clock output is shown in **figure 7-54**.

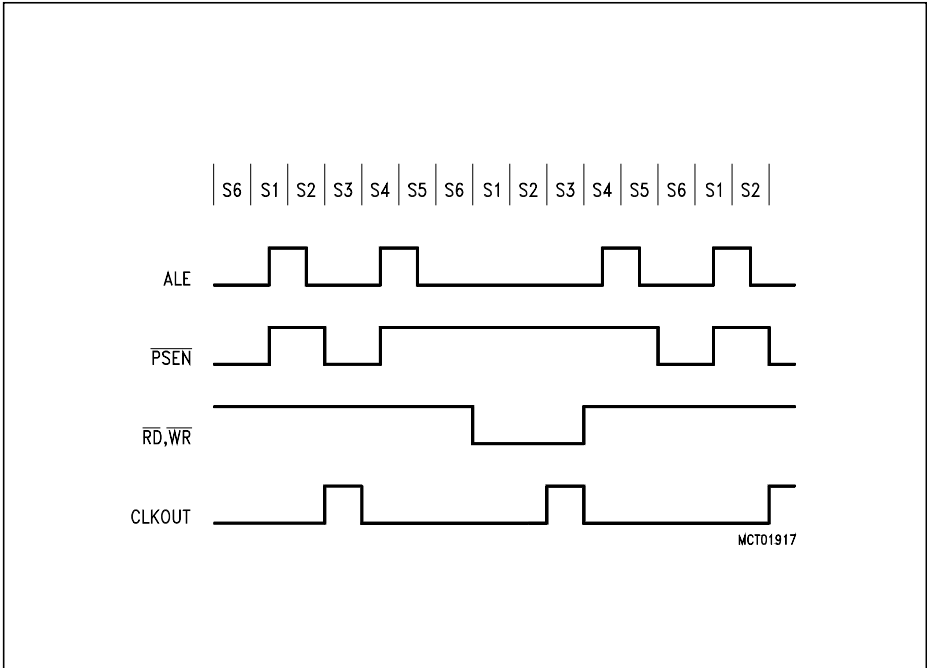


Figure 7-54
Timing Diagram - System Clock Output

8 Interrupt System

The SAB 80C515/80C535 provides 12 interrupt sources with four priority levels.

Five interrupts can be generated by the on-chip peripherals (i.e. timer 0, timer 1, timer 2, compare timer, serial interface and A/D converter), and seven interrupts may be triggered externally (see **figure 8.1**).

8.1 Interrupt Structure

A common mechanism is used to generate the various interrupts, each source having its own request flag(s) located in a special function register (e.g. TCON, IRCON, SCON). Provided that the peripheral or external source meets the condition for an interrupt, the dedicated request flag is set, whether an interrupt is enabled or not. For example, each timer 0 overflow sets the corresponding request flag TF0. If it is already set, it retains a one (1). But the interrupt is not necessarily serviced.

Now each interrupt requested by the corresponding flag can individually be enabled or disabled by the enable bits in SFR's IEN0, IEN1 (see **figure 8-2, 8-3**). This determines whether the interrupt will actually be performed. In addition, there is a global enable bit for all interrupts which, when cleared, disables all interrupts independent of their individual enable bits.

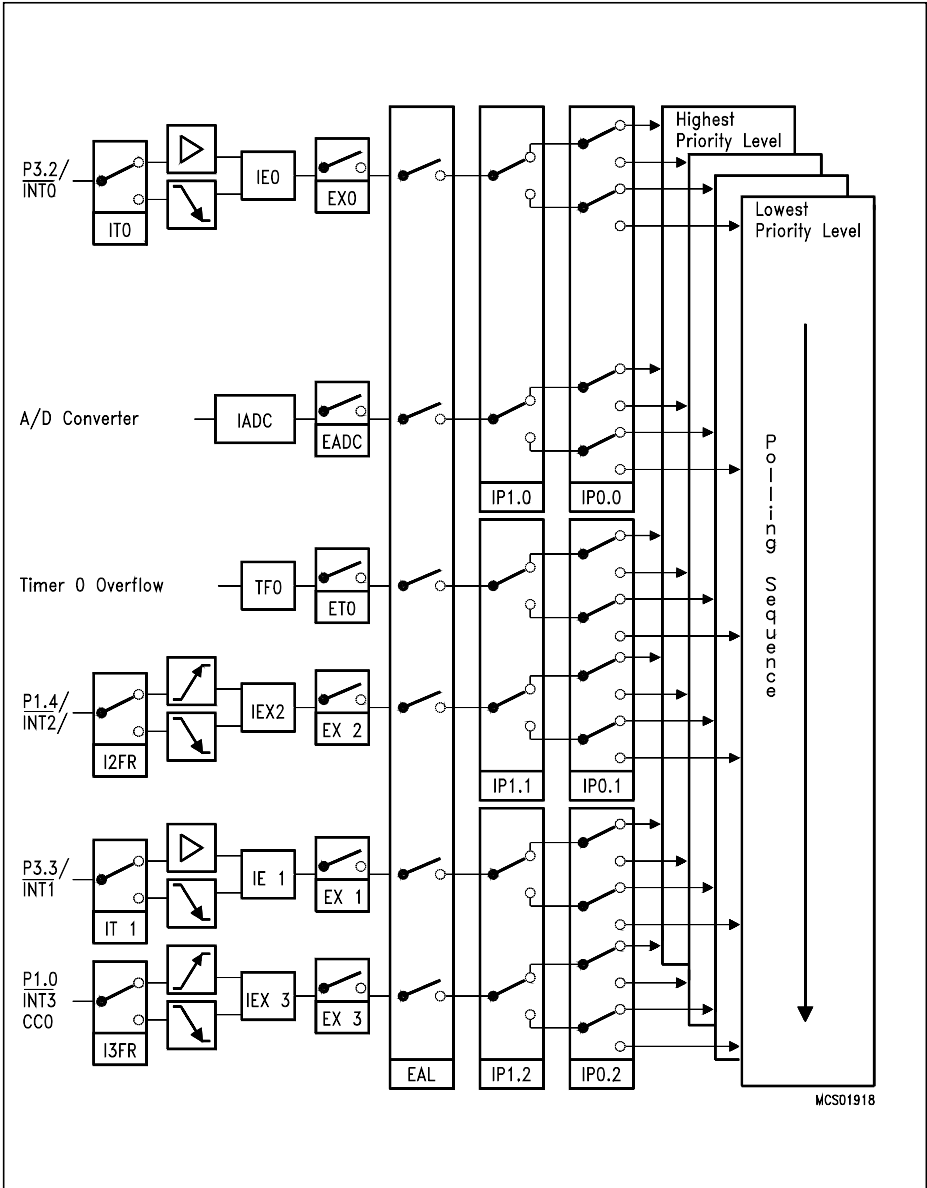


Figure 8-1 a)
Interrupt Structure of the SAB 80(C)515/80(C)535

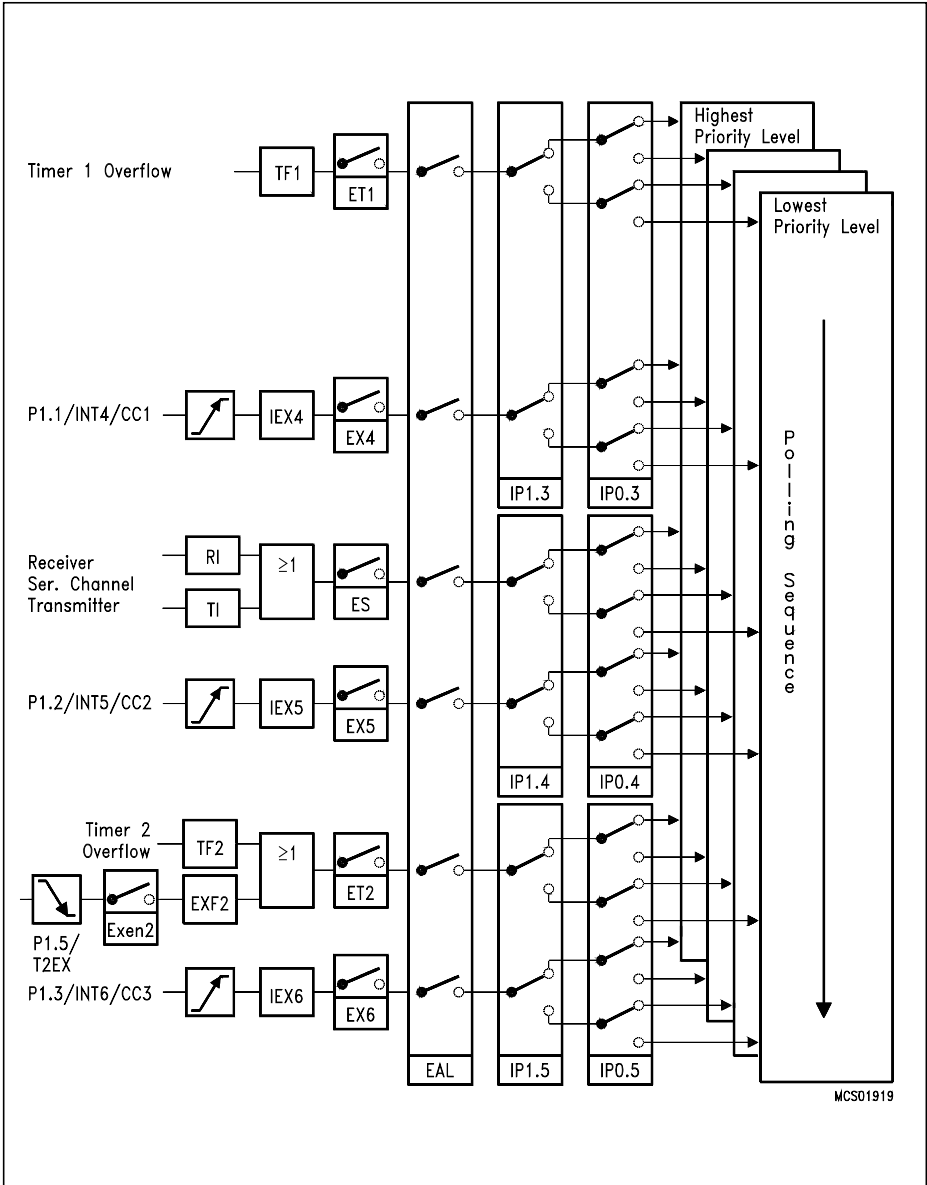
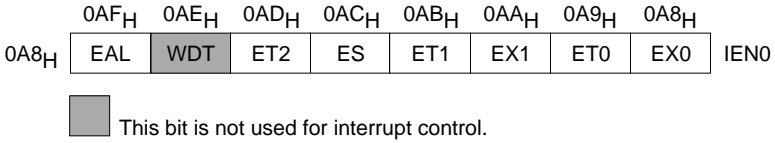


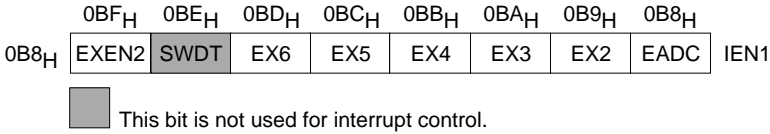
Figure 8-1 b)
Interrupt Structure of the SAB 80(C)515/80(C)535 (cont'd)

Figure 8-2
Special Function Register IEN0 (Address 0A8H)



Bit	Function
EX0	Enables or disables external interrupt 0. If EX0 = 0, external interrupt 0 is disabled.
ET0	Enables or disables the timer 0 overflow interrupt. If ET0 = 0, the timer 0 interrupt is disabled.
EX1	Enables or disables external interrupt 1. If EX1 = 0, external interrupt 1 is disabled.
ET1	Enables or disables the timer 1 overflow interrupt. If ET1 = 0, the timer 1 interrupt is disabled.
ES	Enables or disables the serial channel interrupt. If ES = 0, the serial channel interrupt is disabled.
ET2	Enables or disables the timer 2 overflow or external reload interrupt. If ET2 = 0, the timer 2 interrupt is disabled.
EAL	Enables or disables all interrupts. If EAL = 0, no interrupt will be acknowledged. If EAL = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

Figure 8-3
Special Function Register IEN1 (Address 0B8H)



Bit	Function
EADC	Enables or disables the A/D converter interrupt. If EADC = 0, the A/D converter interrupt is disabled.
EX2	Enables or disables external interrupt 2/capture/compare interrupt 4. If EX2 = 0, external interrupt 2 is disabled.
EX3	Enables or disables external interrupt 3/capture/compare interrupt 0. If EX3 = 0, external interrupt 3 is disabled.
EX4	Enables or disables external interrupt 4/capture/compare interrupt 0. If EX4 = 0, external interrupt 4 is disabled.
EX5	Enables or disables external interrupt 5/capture/compare interrupt 0. If EX5 = 0, external interrupt 5 is disabled.
EX6	Enables or disables external interrupt 6/capture/compare interrupt 0. If EX6 = 0, external interrupt 6 is disabled.
EXEN2	Enables or disables the timer 2 external reload interrupt. EXEN2 = 0 disables the timer 2 external reload interrupt. The external reload function is not affected by EXEN2.

In the following the interrupt sources are discussed individually.

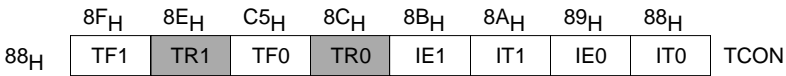
The external interrupts 0 and 1 ($\overline{INT0}$ and $\overline{INT1}$) can each be either level-activated or negative transition-activated, depending on bits IT0 and IT1 in register TCON (see figure 8-4). The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated this interrupt is cleared by the hardware when the service routine is vectored too, but only if the interrupt was transition-activated. If the interrupt was level-activated, then the requesting external source directly controls the request flag, rather than the on-chip hardware.

The timer 0 and timer 1 interrupts are generated by TF0 and TF1 in register TCON, which are set by a rollover in their respective timer/counter registers (exception see section 7.3.4 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 too.

The **serial port interrupt** is generated by a logical OR of flag RI and TI in SFR SCON (see figure 7-7). Neither of these flags is cleared by hardware when the service routine is vectored too. In fact, the service routine will normally have to determine whether it was the receive interrupt flag or the transmission interrupt flag that generated the interrupt, and the bit will have to be cleared by software.

The **timer 2 interrupt** is generated by the logical OR of bit TF2 in register T2CON and bit EXF2 in register IRCON. Figures 8-5 and 8-6 show SFR's T2CON and IRCON. 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 by software.

Figure 8-4
Special Function Register TCON (Address 88H)



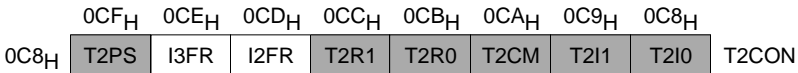
These bits are not used for interrupt control.


Bit	Function
IT0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupts.
IE0	Interrupt 0 edge flag. Set by hardware when external interrupt edge is detected. Cleared when interrupt processed.
IT1	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupts.
IE1	Interrupt 1 edge flag. Set by hardware when external interrupt edge is detected. Cleared when interrupt processed.
TF0	Timer 0 overflow flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.
TF1	Timer 1 overflow flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.

The **A/D converter interrupt** is generated by IADC in register IRCON (see figure 8-6). It is set some cycles before the result is available. That is, if an interrupt is generated, in any case the converted result in ADDAT is valid on the first instruction of the interrupt service routine (with respect to the minimal interrupt response time). If continuous conversions are established, IADC is set once during each conversion. If an A/D converter interrupt is generated, flag IADC will have to be cleared by software.

The external interrupt 2 ($\overline{\text{INT2}}$) can be either positive or negative transition-activated depending on bit I2FR in register T2CON (see figure 8-5). The flag that actually generates this interrupt is bit IEX2 in register IRCON. If an interrupt 2 is generated, flag IEX2 is cleared by hardware when the service routine is vectored too.

Figure 8-5
Special Function Register T2CON (Address 0C8H)



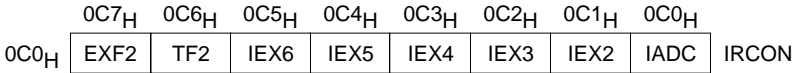
 These bits are not used for interrupt control.

Bit	Function
I2FR	External interrupt 2 falling/rising edge flag. When set, the interrupt 2 request flag IEX2 will be set on a positive transition at pin P1.4/ $\overline{\text{INT2}}$. I2FR = 0 specifies external interrupt 2 to be negative-transition activated.
I3FR	External interrupt 3 falling/rising edge flag. When set, the interrupt 3 request flag IEX3 will be set on a positive transition at pin P1.0/ $\overline{\text{INT3}}$. I3FR = 0 specifies external interrupt 3 to be negative-transition active.

Like the external interrupt 2, the external interrupt 3 ($\overline{\text{INT3}}$) can be either positive or negative transition-activated, depending on bit I3FR in register T2CON. The flag that actually generates this interrupt is bit IEX3 in register IRCON. In addition, this flag will be set if a compare event occurs at pin P1.0/ $\overline{\text{INT3}}$ /CC0, regardless of the compare mode established and the transition at the respective pin. The flag IEX3 is cleared by hardware when the service routine is vectored too.

The external interrupts 4 (INT4), 5 (INT5), 6 (INT6) are positive transition-activated. The flags that actually generate these interrupts are bits IEX4, IEX5, and IEX6 in register IRCON (see figure 8-6). In addition, these flags will be set if a compare event occurs at the corresponding output pin P1.1/INT4/CC1, P1.2/INT5/CC2, and P1.3/INT6/CC3, regardless of the compare mode established and the transition at the respective pin. When an interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored too.

Figure 8-6
Special Function Register IRCON (Address 0C0_H)



Bit	Function
IADC	A/D converter interrupt request flag. Set by hardware at the end of a conversion. Must be cleared by software.
IEX2	External interrupt 2 edge flag. Set by hardware when external interrupt edge was detected or when a compare event occurred at pin 1.4/ $\overline{INT2}$ /CC4. Cleared when interrupt processed.
IEX3	External interrupt 3 edge flag. Set by hardware when external interrupt edge was detected or when a compare event occurred at pin 1.0/ $\overline{INT3}$ /CC0. Cleared when interrupt processed.
IEX4	External interrupt 4 edge flag. Set by hardware when external interrupt edge was detected or when a compare event occurred at pin 1.1/ $\overline{INT4}$ /CC1. Cleared when interrupt processed.
IEX5	External interrupt 5 edge flag. Set by hardware when external interrupt edge was detected or when a compare event occurred at pin 1.2/ $\overline{INT5}$ /CC2. Cleared when interrupt processed.
IEX6	External interrupt 6 edge flag. Set by hardware when external interrupt edge was detected or when a compare event occurred at pin 1.3/ $\overline{INT6}$ /CC3. Cleared when interrupt processed.
TF2	Timer 2 overflow flag. Set by timer 2 overflow. Must be cleared by software. If the timer 2 interrupt is enabled, TF2 = 1 will cause an interrupt.
EXF2	Timer 2 external reload flag. Set when a reload is caused by a negative transition on pin T2EX while EXEN2 = 1. When the timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector the timer 2 interrupt routine. Can be used as an additional external interrupt when the reload function is not used. EXF2 must be cleared by software.

All of these bits that generate interrupts can be set or cleared by software, with the same result as if they had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be cancelled by software. The only exceptions are the request flags IE0 and IE1. If the external interrupts 0 and 1 are programmed to be level-activated, IE0 and IE1 are controlled by the external source via pin $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$, respectively. Thus, writing a one to these bits will not set the request flag IE0 and/or IE1. In this mode, interrupts 0 and 1 can only be generated by software and by writing a 0 to the corresponding pins $\overline{\text{INT0}}$ (P3.2) and $\overline{\text{INT1}}$ (P3.3), provided that this will not affect any peripheral circuit connected to the pins.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in the special function registers IEN0 and IEN1 (**figures 8-2 and 8-3**). Note that IEN0 contains also a global disable bit, EAL, which disables all interrupts at once. Also note that in the SAB 8051 the interrupt priority register IP is located at address 0B8_H; in the SAB 80(C)515/80(C)535 this location is occupied by register IEN1.

8.2 Priority Level Structure

As already mentioned above, all interrupt sources are combined as pairs;

table 8-1 lists the structure of the interrupt sources.

Table 8-1
Pairs of Interrupt Sources

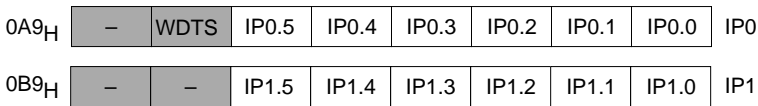
External Interrupt 0	A/D Converter Interrupt
Timer 0 interrupt	External interrupt 2
External interrupt 1	External interrupt 3
Timer 1 interrupt	External interrupt 4
Serial channel 0 interrupt	External interrupt 5
Timer 2 interrupt	External interrupt 6

Each pair of interrupt sources can be programmed individually to one of four priority levels by setting or clearing one bit in the special function register IP0 and one in IP1 (**figure 8-7**). A low-priority interrupt can be interrupted by a high-priority interrupt, but not by another interrupt of the same or a lower priority. An interrupt of the highest priority level cannot be interrupted by another interrupt source.

If two or more requests of different priority levels are received simultaneously, the request of the highest priority is serviced first. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is to be serviced first. Thus, within each priority level there is a second priority structure determined by the polling sequence, as follows (see figure 8-8):

- Within one pair the "left" interrupt is serviced first
- The pairs are serviced from top to bottom of the table.

Figure 8-7
Special Function Registers IP0 and IP1 (Address 0A9_H and 0B9_H)



These bits are not used for interrupt control.

Corresponding bit locations in both registers are used to set the interrupt priority level of an interrupt pair.

Bit		Function
IP1.x	IP0.x	-
0	0	Set priority level 0 (lowest)
0	1	Set priority level 1
1	0	Set priority level 2
1	1	Set priority level 3 (highest)

Bit	Function
IP1.0/IP0.0	IE0/IADC
IP1.1/IP0.1	TF0/IEX2
IP1.2/IP0.2	IE1/IEX3
IP1.3/IP0.3	TF1/IEX4
IP1.4/IP0.4	RI + TI/IEX5
IP1.5/IP0.5	TF2 + EXF2/IEX6

**Figure 8-8
Priority-Within-Level Structure**

High	→	Low	Priority
Interrupt source			
IE0		IADC	High
TF0		IEX2	
IE1		IEX3	↓
TF1		IEX4	
RI + TI		IEX5	
TF2 + EXF2		IEX6	Low

Note:

This "priority-within-level" structure is only used to resolve simultaneous requests of the same priority level.

8.3 How Interrupts are Handled

The interrupt flags are sampled at S5P2 in each machine cycle. The sampled flags 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 a 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 is already in progress.
- 2) The current (polling) cycle is not in the final cycle of the instruction in progress.
- 3) The instruction in progress is RETI or any write access to registers IEN0, IEN1, IEN2 or IP0 and IP1.

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 is completed before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any write access to registers IEN0, IEN1 or IP0 and IP1, then at least one more instruction will be executed before any interrupt is vectored too; this delay guarantees that changes of the interrupt status can be observed by the CPU.

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 that if any interrupt flag is active but not being responded to for one of the conditions already mentioned, or if the flag is no longer 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 interrogates only the pending interrupt requests.

The polling cycle/LCALL sequence is illustrated in **figure 8-9**.

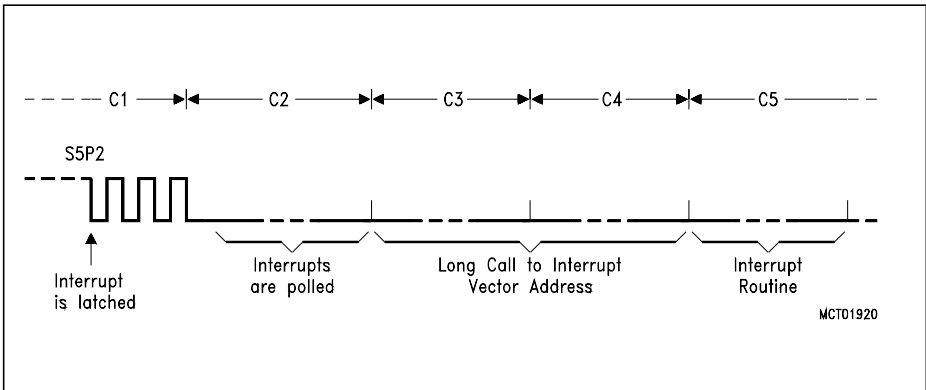


Figure 8-9
Interrupt Response Timing Diagram

Note that if an interrupt of a higher priority level goes active prior to S5P2 in the machine cycle labeled C3 in **figure 8-9**, then, in accordance with the above rules, it will be vectored too during C5 and C6 without any instruction for the lower priority routine to be executed.

Thus, the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, while in other cases it does not; then this has to be done by the user's software. The hardware clears the external interrupt flags IE0 and IE1 only if they were 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 program counter with an address that depends on the source of the interrupt being vectored too, as shown in the following (**table 8-2**).

Table 8-2
Interrupt Sources and Vectors

Interrupt Request Flags	Interrupt Vector Address	Interrupt Source
IE0	0003 _H	External interrupt 0
TF0	000B _H	Timer overflow
IE1	0013 _H	External interrupt 1
TF1	001B _H	Timer 1 overflow
RI/TI	0023 _H	Serial channel
TF2/EXF2	002B _H	Timer 2 overflow/ext. reload
IADC	0043 _H	A/D converter
IEX2	004B _H	External interrupt 2
IEX3	0053 _H	External interrupt 3
IEX4	005B _H	External interrupt 4
IEX5	0063 _H	External interrupt 5
IEX6	006B _H	External interrupt 6

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that the interrupt routine is no longer in progress, then pops the two top bytes from the stack and reloads the program counter. Execution of the interrupted program continues from the point where it was stopped. Note that the RETI instruction is very important because it informs the processor that the program left the current interrupt priority level. 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. In this case no interrupt of the same or lower priority level would be acknowledged.

8.4 External Interrupts

The external interrupts 0 and 1 can be programmed to be level-activated or negative-transition activated by setting or clearing bit IT0 or IT1, respectively, in register TCON (see figure 8-4). If $ITx = 0$ ($x = 0$ or 1), external interrupt x is triggered by a detected low level at the \overline{INTx} pin. If $ITx = 1$, external interrupt x is negative edge-triggered. In this mode, if successive samples of the \overline{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.

If the external interrupt 0 or 1 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.

The external interrupts 2 and 3 can be programmed to be negative or positive transition-activated by setting or clearing bit I2FR or I3FR in register T2CON (see figure 8-5). If $IxFR = 0$ ($x = 2$ or 3), external interrupt x is negative transition-activated. If $IxFR = 1$, external interrupt is triggered by a positive transition.

The external interrupts 4, 5, and 6 are activated by a positive transition. The external timer 2 reload trigger interrupt request flag EXF2 will be activated by a negative transition at pin P1.5/T2EX but only if bit EXEN2 is set.

Since the external interrupt pins ($\overline{INT2}$ to INT6) are sampled once in each machine cycle, an input high or low should be held 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 low (high for $\overline{INT2}$ and $\overline{INT3}$, if it is programmed to be negative transition-active) for at least one cycle, and then hold it high (low) for at least one cycle to ensure that the transition is recognized so that the corresponding interrupt request flag will be set (see figure 8-10). The external interrupt request flags will automatically be cleared by the CPU when the service routine is called.

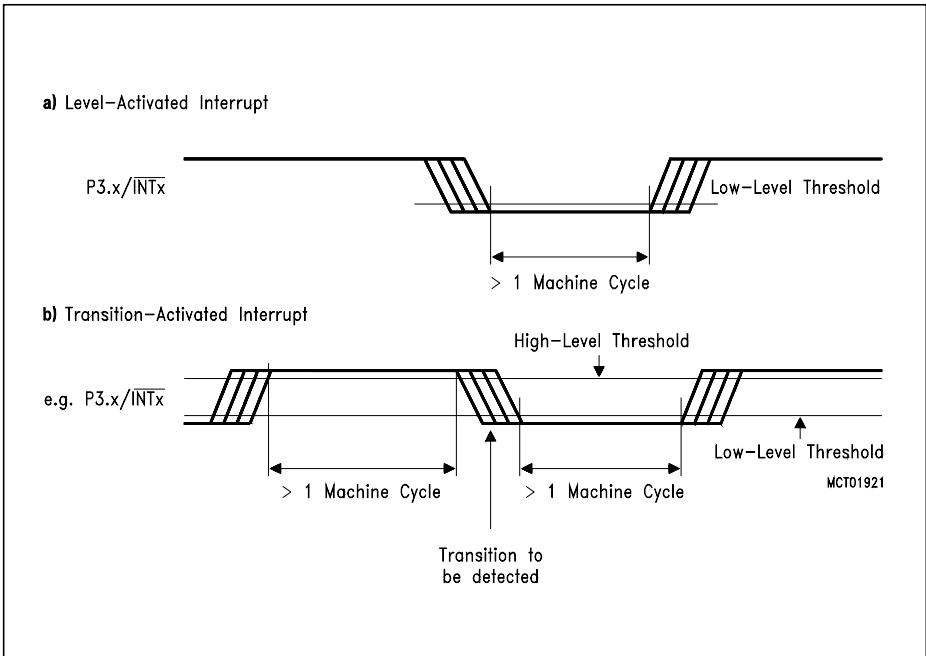


Figure 8-10
External Interrupt Detection

8.5 Response Time

If an external interrupt is recognized, its corresponding request flag is set at S5P2 in every machine cycle. The value is not polled by the circuitry until the next machine cycle. If the 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 will elapse between activation and external interrupt request and the beginning of execution of the first instruction of the service routine.

A longer response time would be obtained if the request was blocked by one of the three previously listed conditions. If an interrupt of equal or higher priority 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 a write access to registers IEN0, IEN1 or IP0, IP1, 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.

9 Instruction Set

The SAB 80(C)515/80(C)535 instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three-byte instructions. The instruction opcode format consists of a function mnemonic followed by a "destination, source" operand field. This field specifies the data type and addressing method(s) to be used.

Like all other members of the 8051-family, the SAB 80(C)515/80(C)535 can be programmed with the same instruction set common to the basic member, the SAB 8051.

Thus, the SAB 80(C)515/80(C)535 is 100% software compatible to the SAB 8051 and may be programmed with 8051 assembler or high-level languages.

9.1 Addressing Modes

The SAB 80(C)515/80(C)535 uses five addressing modes:

- register
- direct
- immediate
- register indirect
- base register plus index-register indirect

Table 9-1 summarizes the memory spaces which may be accessed by each of the addressing modes.

Register Addressing

Register addressing accesses the eight working registers (R0 - R7) of the selected register bank. The least significant bit of the instruction opcode indicates which register is to be used. ACC, B, DPTR and CY, the Boolean processor accumulator, can also be addressed as registers.

Direct Addressing

Direct addressing is the only method of accessing the special function registers. The lower 128 bytes of internal RAM are also directly addressable.

Immediate Addressing

Immediate addressing allows constants to be part of the instruction in program memory.

**Table 9-1
Addressing Modes and Associated Memory Spaces**

Addressing Modes	Associated Memory Spaces
Register addressing	R0 through R7 of selected register bank, ACC, B, CY (Bit), DPTR
Direct addressing	Lower 128 bytes of internal RAM, special function registers
Immediate addressing	Program memory
Register indirect addressing	Internal RAM (@R1, @R0, SP), external data memory (@R1, @R0, @DPTR)
Base register plus index register addressing	Program memory (@DPTR + A, @PC + A)

Register Indirect Addressing

Register indirect addressing uses the contents of either R0 or R1 (in the selected register bank) as a pointer to locations in a 256-byte block: the 256 bytes of internal RAM or the lower 256 bytes of external data memory. Note that the special function registers are not accessible by this method. The upper half of the internal RAM can be accessed by indirect addressing only. Access to the full 64 Kbytes of external data memory address space is accomplished by using the 16-bit data pointer. Execution of PUSH and POP instructions also uses register indirect addressing. The stack may reside anywhere in the internal RAM.

Base Register plus Index Register Addressing

Base register plus index register addressing allows a byte to be accessed from program memory via an indirect move from the location whose address is the sum of a base register (DPTR or PC) and index register, ACC. This mode facilitates look-up table accesses.

Boolean Processor

The Boolean processor is a bit processor integrated into the SAB 80(C)515/80(C)535. It has its own instruction set, accumulator (the carry flag), bit-addressable RAM and I/O.

The Bit Manipulation Instructions allow:

- set bit
- clear bit
- complement bit
- jump if bit is set
- jump if bit is not set
- jump if bit is set and clear bit
- move bit from / to carry

Addressable bits, or their complements, may be logically AND-ed or OR-ed with the contents of the carry flag. The result is returned to the carry register.

9.2 Introduction to the Instruction Set

The instruction set is divided into four functional groups:

- data transfer
- arithmetic
- logic
- control transfer

9.2.1 Data Transfer

Data operations are divided into three classes:

- general-purpose
- accumulator-specific
- address-object

None of these operations affects the PSW flag settings except a POP or MOV directly to the PSW.

General-Purpose Transfers

- MOV performs a bit or byte transfer from the source operand to the destination operand.
- PUSH increments the SP register and then transfers a byte from the source operand to the stack location currently addressed by SP.
- POP transfers a byte operand from the stack location addressed by the SP to the destination operand and then decrements SP.

Accumulator-Specific Transfers

- XCH exchanges the byte source operand with register A (accumulator).
- XCHD exchanges the low-order nibble of the source operand byte with the low-order nibble of A.
- MOVX performs a byte move between the external data memory and the accumulator. The external address can be specified by the DPTR register (16 bit) or the R1 or R0 register (8 bit).
- MOVC moves a byte from program memory to the accumulator. The operand in A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to the accumulator.

Address-Object Transfer

- MOV DPTR, #data loads 16 bits of immediate data into a pair of destination registers, DPH and DPL.

9.2.2 Arithmetic

The SAB 80(C)515/80(C)535 has four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed BCD representations.

Addition

- INC (increment) adds one to the source operand and puts the result in the operand.
- ADD adds A to the source operand and returns the result to A.
- ADDC (add with carry) adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.
- DA (decimal-add-adjust for BCD addition) corrects the sum which results from the binary addition of two-digit decimal operands. The packed decimal sum formed by DA is returned to A. CY is set if the BCD result is greater than 99; otherwise, it is cleared.

Subtraction

- SUBB (subtract with borrow) subtracts the second source operand from the the first operand (the accumulator), subtracts one (1) if CY is set and returns the result to A.
- DEC (decrement) subtracts one (1) from the source operand and returns the result to the operand.

Multiplication

- MUL performs an unsigned multiplication of the A register, returning a double byte result. A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is not zero. CY is cleared. AC is unaffected.

Division

- DIV performs an unsigned division of the A register by the B register; it returns the integer quotient to the A register and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV; otherwise, OV is cleared. CY is cleared. AC remains unaffected.

Flags

Unless otherwise stated in the previous descriptions, the flags of PSW are affected as follows:

- CY is set if the operation causes a carry to or a borrow from the resulting high-order bit; otherwise CY is cleared.
- AC is set if the operation results in a carry from the low-order four bits of the result (during addition), or a borrow from the high-order bits to the low-order bits (during subtraction); otherwise AC is cleared.
- OV is set if the operation results in a carry to the high-order bit of the result but not a carry from the bit, or vice versa; otherwise OV is cleared. OV is used in two's-complement arithmetic, because it is set when the signal result cannot be represented in 8 bits.
- P is set if the modulo-2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

9.2.3 Logic

The SAB 80(C)515/80(C)535 performs basic logic operations on both bit and byte operands.

Single-Operand Operations

- CLR sets A or any directly addressable bit to zero (0).
- SETB sets any directly bit-addressable bit to one (1).
- CPL is used to complement the contents of the A register without affecting any flag, or any directly addressable bit location.
- RL, RLC, RR, RRC, SWAP are the five operations that can be performed on A. RL, rotate left, RR, rotate right, RLC, rotate left through carry, RRC, rotate right through carry, and SWAP, rotate left four. For RLC and RRC the CY flag becomes equal to the last bit rotated out. SWAP rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

Two-Operand Operations

- ANL performs bitwise logical AND of two operands (for both bit and byte operands) and returns the result to the location of the first operand.
- ORL performs bitwise logical OR of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- XRL performs logical Exclusive OR of two source operands (byte operands) and returns the result to the location of the first operand.

9.2.4 Control Transfer

There are three classes of control transfer operations: unconditional calls, returns, jumps, conditional jumps, and interrupts. All control transfer operations, some upon a specific condition, cause the program execution to continue a non-sequential location in program memory.

Unconditional Calls, Returns and Jumps

Unconditional calls, returns and jumps transfer control from the current value of the program counter to the target address. Both direct and indirect transfers are supported.

- ACALL and LCALL push the address of the next instruction onto the stack and then transfer control to the target address. ACALL is a 2-byte instruction used when the target address is in the current 2K page. LCALL is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. an 11-bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.
- RET transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.
- AJMP, LJMP and SJMP transfer control to the target operand. The operation of AJMP and LJMP are analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256-byte range centered about the starting address of the next instruction (-128 to $+127$).
- JMP @A + DPTR performs a jump relative to the DPTR register. The operand in A is used as the offset ($0 - 255$) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the program memory space.

Conditional Jumps

Conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (-128 to $+127$).

- JZ performs a jump if the accumulator is zero.
- JNZ performs a jump if the accumulator is not zero.
- JC performs a jump if the carry flag is set.
- JNC performs a jump if the carry flag is not set.
- JB performs a jump if the directly addressed bit is set.
- JNB performs a jump if the directly addressed bit is not set.
- JBC performs a jump if the directly addressed bit is set and then clears the directly addressed bit.
- CJNE compares the first operand to the second operand and performs a jump if they are not equal. CY is set if the first operand is less than the second operand; otherwise it is cleared. Comparisons can be made between A and directly addressable bytes in internal data memory or an immediate value and either A, a register in the selected register bank, or a register indirectly addressable byte of the internal RAM.
- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The source operand of the DJNZ instruction may be any directly addressable byte in the internal data memory. Either direct or register addressing may be used to address the source operand.

Interrupt Returns

- RETI transfers control as RET does, but additionally enables interrupts of the current priority level.

9.3 Instruction Definitions

All 111 instructions of the SAB 80(C)515/80(C)535 can essentially be condensed to 54 basic operations, in the following alphabetically ordered according to the operation mnemonic section.

Instruction	Flag			Instruction	Flag		
	CY	OV	AC		CY	OV	AC
ADD	X	X	X	SETB C	1		
ADDC	X	X	X	CLR C	0		
SUBB	X	X	X	CPL C	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ANL C,/bit	X		
DA	X			ORL C,/bit	X		
RRC	X			ORL C,/bit	X		
RLC	X			MOV C,/bit	X		
CJNE	X						

A brief example of how the instruction might be used is given as well as its effect on the PSW flags. The number of bytes and machine cycles required, the binary machine language encoding, and a symbolic description or restatement of the function is also provided.

Note:

Only the carry, auxiliary carry, and overflow flags are discussed. The parity bit is computed after every instruction cycle that alters the accumulator.

Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit manipulation.

Notes on Data Addressing Modes

- Rn - Working register R0-R7
- direct - 128 internal RAM locations, any I/O port, control or status register
- @Ri - Indirect internal or external RAM location addressed by register R0 or R1
- #data - 8-bit constant included in instruction
- #data 16 - 16-bit constant included as bytes 2 and 3 of instruction
- bit - 128 software flags, any bitaddressable I/O pin, control or status bit
- A - Accumulator

Notes on Program Addressing Modes

- addr16 - Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte program memory address space.
- addr11 - Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.
- rel - SJMP and all conditional jumps include an 8 bit offset byte. Range is + 127/- 128 bytes relative to the first byte of the following instruction.

All mnemonics copyrighted: © Intel Corporation 1980

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, op code bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of 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 0345H. After executing the instruction

```
ACALL SUBRTN
```

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

Operation: ACALL
 $(PC) \leftarrow (PC) + 2$
 $(SP) \leftarrow (SP) + 1$
 $((SP)) \leftarrow (PC7-0)$
 $(SP) \leftarrow (SP) + 1$
 $((SP)) \leftarrow (PC15-8)$
 $(PC10-0) \leftarrow \text{page address}$

Encoding:

a10	a9	a8	1	0	0	0	1
-----	----	----	---	---	---	---	---

a7	a6	a5	a4	a3	a2	a1	a0
----	----	----	----	----	----	----	----

Bytes: 2

Cycles: 2

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 of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

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 0C3_H (11000011_B) and register 0 holds 0AA_H (10101010_B).
The instruction

ADD A,R0

will leave 6D_H (01101101_B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Operation: ADD
(A) ← (A) + (Rn)

Encoding:

0 0 1 0	1 r r r
---------	---------

Bytes: 1

Cycles: 1

ADD A,direct

Operation: ADD
(A) ← (A) + (direct)

Encoding:

0 0 1 0	0 1 0 1	direct address
---------	---------	----------------

Bytes: 2

Cycles: 1

ADD A, @Ri

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

Encoding:

0 0 1 0	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

ADD A, #data

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

Encoding:

0 0 1 0	0 1 0 0	immediate data
---------	---------	----------------

Bytes: 2

Cycles: 1

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 of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

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 0C3_H (11000011B) and register 0 holds 0AA_H (10101010B) with the carry flag set. The instruction

ADDC A,R0

will leave 6E_H (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

ADDC A,Rn

Operation: ADDC
 (A) ← (A) + (C) + (Rn)

Encoding:

0	0	1	1	1	r	r	r
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

ADDC A,direct

Operation: ADDC
 (A) ← (A) + (C) + (direct)

Encoding:

0	0	1	1
---	---	---	---

0	1	0	1
---	---	---	---

direct address

Bytes: 2

Cycles: 1

ADDC A, @Ri

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

Encoding:

0 0 1 1	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

ADDC A, #data

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

Encoding:

0 0 1 1	0 1 0 0
---------	---------

immediate data

Bytes: 2

Cycles: 1

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), op code 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 0123_H. The instruction
AJMP JMPADR
is at location 0345_H and will load the PC with 0123_H.

Operation: AJM P
(PC) ← (PC) + 2
(PC10-0) ← page address

Encoding:

a10	a9	a8	0	0	0	1
-----	----	----	---	---	---	---

a7	a6	a5	a4	a3	a2	a1	a0
----	----	----	----	----	----	----	----

Bytes: 2

Cycles: 2

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 a 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 0C3_H (11000011B) and register 0 holds 0AA_H (10101010B) then the instruction

ANL A,R0

will leave 81_H (10000001B) 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

Operation: ANL
 (A) ← (A) ∧ (Rn)

Encoding:

0 1 0 1	1 r r r
---------	---------

Bytes: 1

Cycles: 1

ANL A, direct

Operation: ANL
 $(A) \leftarrow (A) \wedge (\text{direct})$

Encoding:

0 1 0 1	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

ANL A, @Ri

Operation: ANL
 $(A) \leftarrow (A) \wedge ((Ri))$

Encoding:

0 1 0 1	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

ANL A, #data

Operation: ANL
 $(A) \leftarrow (A) \wedge \#data$

Encoding:

0 1 0 1	0 1 0 0
---------	---------

immediate data

Bytes: 2

Cycles: 1

ANL direct, A

Operation: ANL
 $(\text{direct}) \leftarrow (\text{direct}) \wedge (A)$

Encoding:

0 1 0 1	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

ANL **direct, #data**

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

Encoding:

0	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

direct address

immediate data

Bytes: 3

Cycles: 2

ANL C, <src-bit>

Function: Logical AND for bit variables

Description: If the Boolean value of the source bit is a logic 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 bit 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 accumulator bit 7
ANL    C,/OV            ; AND with inverse of overflow flag
```

ANL C,bit

Operation: ANL
 $(C) \leftarrow (C) \wedge (\text{bit})$

Encoding:

1 0 0 0	0 0 1 0
---------	---------

bit address

Bytes: 2

Cycles: 2

ANL C,/bit

Operation: ANL
 $(C) \leftarrow (C) \wedge \neg (\text{bit})$

Encoding:

1 0 1 1	0 0 0 0
---------	---------

bit address

Bytes: 2

Cycles: 2

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 34_H. Register 7 contains 56_H. 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 60_H.

If the data being presented to port 1 is also 34_H, 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 input on P1, the program will loop at this point until the P1 data changes to 34_H).

CJNE A, direct, rel

Operation: $(PC) \leftarrow (PC) + 3$
 if $(A) < > (\text{direct})$
 then $(PC) \leftarrow (PC) + \text{relative offset}$
 if $(A) < (\text{direct})$
 then $(C) \leftarrow 1$
 else $(C) \leftarrow 0$

Encoding:

1 0 1 1	0 1 0 1
---------	---------

direct address

rel. address

Bytes: 3

Cycles: 2

CJNE A, #data, rel

Operation: $(PC) \leftarrow (PC) + 3$
 if $(A) < > \text{data}$
 then $(PC) \leftarrow (PC) + \text{relative offset}$
 if $(A) \leftarrow \text{data}$
 then $(C) \leftarrow 1$
 else $(C) \leftarrow 0$

Encoding:

1 0 1 1	0 1 0 0
---------	---------

immediate data

rel. address

Bytes: 3

Cycles: 2

CJNE RN, #data, rel

Operation: $(PC) \leftarrow (PC) + 3$
 if $(Rn) < > \text{data}$
 then $(PC) \leftarrow (PC) + \text{relative offset}$
 if $(Rn) < \text{data}$
 then $(C) \leftarrow 1$
 else $(C) \leftarrow 0$

Encoding:

1 0 1 1	1 r r r
---------	---------

immediate data

rel. address

Bytes: 3

Cycles: 2

CJNE @Ri, #data,rel

Operation: $(PC) \leftarrow (PC) + 3$
 if $((Ri) < > \text{data})$
 then $(PC) \leftarrow (PC) + \text{relative offset}$
 if $((Ri) < \text{data})$
 then $(C) \leftarrow 1$
 else $(C) \leftarrow 0$

Encoding:

1	0	1	1
---	---	---	---

0	1	1	i
---	---	---	---

immediate data

rel. address

Bytes: 3

Cycles: 2

CLR A

Function: Clear accumulator

Description: The accumulator is cleared (all bits set to zero). No flags are affected.

Example: The accumulator contains 5C_H (01011100_B). The instruction
CLR A
will leave the accumulator set to 00_H (00000000_B).

Operation: CLR
 (A) ← 0

Encoding:

1	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 5D_H (01011101B). The instruction
 CLR P1.2
 will leave the port set to 59_H (01011001B).

CLR C

Operation: CLR
 (C) ← 0

Encoding:

1	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

CLR bit

Operation: CLR
 (bit) ← 0

Encoding:

1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

bit address

Bytes: 2

Cycles: 1

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 zero and vice versa. No flags are affected.

Example: The accumulator contains 5C_H (01011100_B). The instruction

 CPL A

 will leave the accumulator set to 0A3_H (10100011_B).

Operation: CPL

 (A) ← ¬ (A)

Encoding:

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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. CPL can operate on the carry or any directly addressable 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 5D_H (01011101_B). The instruction sequence

CPL P1.1

CPL P1.2

will leave the port set to 5B_H (01011011_B).

CPL C

Operation: CPL
(C) ← ¬(C)

Encoding:

1 0 1 1	0 0 1 1
---------	---------

Bytes: 1

Cycles: 1

CPL bit

Operation: CPL
(bit) ← ¬(bit)

Encoding:

1 0 1 1	0 0 1 0
---------	---------

bit address

Bytes: 2

Cycles: 1

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-1111xxxx), 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 00_H, 06_H, 60_H, or 66_H 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 56_H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67_H (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 two's-complement binary addition, resulting in the value 0BE_H (10111110B) in the accumulator. The carry and auxiliary carry flags will be cleared.

The decimal adjust instruction will then alter the accumulator to the value 24_H (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 01_H or 99_H. If the accumulator initially holds 30_H (representing the digits of 30 decimal), then the instruction sequence

```
ADD    A, #99H
DA     A
```

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

Operation: DA
 contents of accumulator are BCD
 if $[(A3-0) > 9] \vee [(AC) = 1]$
 then $(A3-0) \leftarrow (A3-0) + 6$
 and
 if $[(A7-4) > 9] \vee [(C) = 1]$
 then $(A7-4) \leftarrow (A7-4) + 6$

Encoding:

1 1 0 1	0 1 0 0
---------	---------

Bytes: 1

Cycles: 1

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00_H will underflow to 0FF_H. 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 7F_H (01111111B). Internal RAM locations 7E_H and 7F_H contain 00_H and 40_H, respectively. The instruction sequence

```
DEC     @R0
DEC     R0
DEC     @R0
```

will leave register 0 set to 7E_H and internal RAM locations 7E_H and 7F_H set to 0FF_H and 3F_H.

DEC A

Operation: DEC
 (A) ← (A) – 1

Encoding:

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

DEC Rn

Operation: DEC
 (Rn) ← (Rn) – 1

Encoding:

0	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

DEC direct

Operation: DEC
 $(\text{direct}) \leftarrow (\text{direct}) - 1$

Encoding:

0 0 0 1	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

DEC @Ri

Operation: DEC
 $((\text{Ri})) \leftarrow ((\text{Ri})) - 1$

Encoding:

0 0 0 1	0 1 1 i
---------	---------

Bytes: 1

Cycles: 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 00_H, 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 (0FB_H or 11111011_B) and B contains 18 (12_H or 00010010_B). The instruction

DIV AB

will leave 13 in the accumulator (0D_H or 00001101_B) and the value 17 (11_H or 00010001_B) in B, since 251 = (13x18) + 17. Carry and OV will both be cleared.

Operation: DIV

$$\begin{matrix} (A15-8) \\ (B7-0) \end{matrix} \leftarrow (A) / (B)$$

Encoding:

1	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 4

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 00_H will underflow to 0FF_H. 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 40_H, 50_H, and 60_H contain the values, 01_H, 70_H, and 15_H, 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 00_H, 6F_H, and 15_H 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

Operation: DJNZ
 $(PC) \leftarrow (PC) + 2$
 $(Rn) \leftarrow (Rn) - 1$
 if $(Rn) > 0$ or $(Rn) < 0$
 then $(PC) \leftarrow (PC) + rel$

Encoding:

1 1 0 1	1 r r r
---------	---------

rel. address

Bytes: 2

Cycles: 2

DJNZ direct,rel

Operation: DJNZ
 $(PC) \leftarrow (PC) + 2$
 $(direct) \leftarrow (direct) - 1$
 if $(direct) > 0$ or $(direct) < 0$
 then $(PC) \leftarrow (PC) + rel$

Encoding:

1 1 0 1	0 1 0 1
---------	---------

direct address

rel. address

Bytes: 3

Cycles: 2

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FF_H will overflow to 00_H. 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 7E_H (01111110B). Internal RAM locations 7E_H and 7F_H contain 0FF_H and 40_H, respectively. The instruction sequence

```
INC    @R0
INC    R0
INC    @R0
```

will leave register 0 set to 7F_H and internal RAM locations 7E_H and 7F_H holding (respectively) 00_H and 41_H.

INC A

Operation: INC
 $(A) \leftarrow (A) + 1$

Encoding:

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

INC Rn

Operation: INC
 $(Rn) \leftarrow (Rn) + 1$

Encoding:

0	0	0	0	0	1	r	r	r
---	---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

INC direct

Operation: INC
 $(\text{direct}) \leftarrow (\text{direct}) + 1$

Encoding:

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address

Bytes: 2

Cycles: 1

INC @Ri

Operation: INC
 $((\text{Ri})) \leftarrow ((\text{Ri})) + 1$

Encoding:

0	0	0	0	0	0	1	1	i
---	---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 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 $0FF_H$ to 00_H 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 12_H and $0FE_H$, respectively. The instruction sequence

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

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

Operation: INC
 $(DPTR) \leftarrow (DPTR) + 1$

Encoding:

1 0 1 0	0 0 1 1
---------	---------

Bytes: 1

Cycles: 2

JB **bit,rel**

Function: Jump if bit is 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.

Operation: JB
 $(PC) \leftarrow (PC) + 3$
 if (bit) = 1
 then $(PC) \leftarrow (PC) + rel$

Encoding:

0 0 1 0	0 0 0 0
---------	---------

bit address

rel. address

Bytes: 3

Cycles: 2

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. *In either case, clear the designated bit.* 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 56_H (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 52_H (01010010B).

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

Encoding:

0 0 0 1	0 0 0 0
---------	---------

bit address

rel. address

Bytes: 3

Cycles: 2

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      LABEL2
```

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

Operation: JC
 $(PC) \leftarrow (PC) + 2$
 if $(C) = 1$
 then $(PC) \leftarrow (PC) + rel$

Encoding:

0 1 0 0	0 0 0 0
---------	---------

rel. address

Bytes: 2

Cycles: 2

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 04_{H} 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.

Operation: JMP
 $(PC) \leftarrow (A) + (DPTR)$

Encoding:

0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 2

JNB **bit,rel**

Function: Jump if bit is 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.

Operation: JNB
 (PC) ← (PC) + 3
 if (bit) = 0
 then (PC) ← (PC) + rel.

Encoding:

0 0 1 1	0 0 0 0
---------	---------

bit address

rel. address

Bytes: 3

Cycles: 2

JNC rel

Function: Jump if carry is 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.

Operation: JNC
 $(PC) \leftarrow (PC) + 2$
 if $(C) = 0$
 then $(PC) \leftarrow (PC) + rel$

Encoding:

0 1 0 1	0 0 0 0
---------	---------

rel. address

Bytes: 2

Cycles: 2

JNZ rel

Function: Jump if accumulator is 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 00_H. The instruction sequence

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

will set the accumulator to 01_H and continue at label LABEL2.

Operation: JNZ
 $(PC) \leftarrow (PC) + 2$
 if $(A) \neq 0$
 then $(PC) \leftarrow (PC) + rel.$

Encoding:



Bytes: 2

Cycles: 2

JZ rel

Function: Jump if accumulator is 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 01_H. The instruction sequence

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

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

Operation: JZ
 $(PC) \leftarrow (PC) + 2$
 if $(A) = 0$
 then $(PC) \leftarrow (PC) + rel$

Encoding:

0	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

rel. address

Bytes: 2

Cycles: 2

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 64 Kbyte program memory address space. No flags are affected.

Example: Initially the stack pointer equals 07_H. The label "SUBRTN" is assigned to program memory location 1234_H. After executing the instruction

LCALL SUBRTN

at location 0123_H, the stack pointer will contain 09_H, internal RAM locations 08_H and 09_H will contain 26_H and 01_H, and the PC will contain 1234_H.

Operation: LCALL

(PC) ← (PC) + 3
 (SP) ← (SP) + 1
 ((SP)) ← (PC7-0)
 (SP) ← (SP) + 1
 ((SP)) ← (PC15-8)
 (PC) ← addr15-0

Encoding:

0	0	0	1	0	0	1	0
---	---	---	---	---	---	---	---

addr15 . . . addr8

addr7 . . . addr0

Bytes: 3

Cycles: 2

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 1234_H. The instruction

```
LJMP    JMPADR
```

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

Operation: LJMP
 (PC) ← addr15-0

Encoding:

0 0 0 0	0 0 1 0
---------	---------

addr15 . . . addr8

addr7 . . . addr0

Bytes: 3

Cycles: 2

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 30_H holds 40_H. The value of RAM location 40_H is 10_H. The data present at input port 1 is 11001010B (0CA_H).

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

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

MOV A, Rn

Operation: MOV
(A) ← (Rn)

Encoding:

1	1	1	0	1	r	r	r
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

MOV A, direct *)

Operation: MOV
(A) ← (direct)

Encoding:

1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address

Bytes: 2

Cycles: 1

*) MOV A, ACC is not a valid instruction.

MOV A,@Ri

Operation: MOV
(A) ← ((Ri))

Encoding:

1 1 1 0	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

MOV A,#data

Operation: MOV
(A) ← #data

Encoding:

0 1 1 1	0 1 0 0
---------	---------

immediate data

Bytes: 2

Cycles: 1

MOV Rn,A

Operation: MOV
(Rn) ← (A)

Encoding:

1 1 1 1	1 r r r
---------	---------

Bytes: 1

Cycles: 1

MOV Rn,direct

Operation: MOV
(Rn) ← (direct)

Encoding:

1 0 1 0	1 r r r
---------	---------

direct address

Bytes: 2

Cycles: 2

MOV Rn, #data

Operation: MOV
(Rn) ← #data

Encoding:

0 1 1 1	1 r r r
---------	---------

immediate data

Bytes: 2

Cycles: 1

MOV direct,A

Operation: MOV
(direct) ← (A)

Encoding:

1 1 1 1	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

MOV direct,Rn

Operation: MOV
(direct) ← (Rn)

Encoding:

1 0 0 0	1 r r r
---------	---------

direct address

Bytes: 2

Cycles: 2

MOV direct,direct

Operation: MOV
(direct) ← (direct)

Encoding:

1 0 0 0	0 1 0 1
---------	---------

dir.addr. (src) dir.addr. (dest)

Bytes: 3

Cycles: 2

MOV direct, @ Ri

Operation: MOV
(direct) ← ((Ri))

Encoding:

1 0 0 0	0 1 1 i
---------	---------

direct address

Bytes: 2

Cycles: 2

MOV direct, #data

Operation: MOV
(direct) ← #data

Encoding:

0 1 1 1	0 1 0 1
---------	---------

direct address

immediate data

Bytes: 3

Cycles: 2

MOV @ Ri,A

Operation: MOV
((Ri)) ← (A)

Encoding:

1 1 1 1	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

MOV @ Ri,direct

Operation: MOV
((Ri)) ← (direct)

Encoding:

1 0 1 0	0 1 1 i
---------	---------

direct address

Bytes: 2

Cycles: 2

MOV @ Ri,#data

Operation: MOV
((Ri)) ← #data

Encoding:

0	1	1	1
---	---	---	---

0	1	1	i
---	---	---	---

immediate data

Bytes: 2

Cycles: 1

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 35_H (00110101B).

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

will leave the carry cleared and change port 1 to 39_H (00111001 B).

MOV C,bit

Operation: MOV
(C) ← (bit)

Encoding:

1 0 1 0	0 0 1 0
---------	---------

bit address

Bytes: 2

Cycles: 1

MOV bit,C

Operation: MOV
(bit) ← (C)

Encoding:

1 0 0 1	0 0 1 0
---------	---------

bit address

Bytes: 2

Cycles: 2

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 1234_H into the data pointer: DPH will hold 12_H and DPL will hold 34_H.

Operation: MOV
(DPTR) ← #data15-0
DPH □ DPL ← #data15-8 □ #data7-0

Encoding:

1	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

immed. data 15 . . . 8

immed. data 7 . . . 0

Bytes: 3

Cycles: 2

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 to 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 01_H, it will return with 77_H 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

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

Encoding:

1 0 0 1	0 0 1 1
---------	---------

Bytes: 1

Cycles: 2

MOVC A, @A + PC

Operation: MOVC
 (PC) \leftarrow (PC) + 1
 (A) \leftarrow ((A) + (PC))

Encoding:

1 0 0 0	0 0 1 1
---------	---------

Bytes: 1

Cycles: 2

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 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 instructions, 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 64 Kbyte), 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 SAB 8155 RAM/I/O/timer) is connected to the SAB 80(c)5XX 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 12_H and 34_H. Location 34_H of the external RAM holds the value 56_H. The instruction sequence

```
MOVX  A, @R1
MOVX  @R0,A
```

copies the value 56_H into both the accumulator and external RAM location 12_H.

MOVX A,@Ri

Operation: MOVX
 $(A) \leftarrow ((Ri))$

Encoding:

1 1 1 0	0 0 1 i
---------	---------

Bytes: 1

Cycles: 2

MOVX A,@DPTR

Operation: MOVX
 $(A) \leftarrow ((DPTR))$

Encoding:

1 1 1 0	0 0 0 0
---------	---------

Bytes: 1

Cycles: 2

MOVX @Ri,A

Operation: MOVX
 $((Ri)) \leftarrow (A)$

Encoding:

1 1 1 1	0 0 1 i
---------	---------

Bytes: 1

Cycles: 2

MOVX @DPTR,A

Operation: MOVX
 $((DPTR)) \leftarrow (A)$

Encoding:

1 1 1 1	0 0 0 0
---------	---------

Bytes: 1

Cycles: 2

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 (0FF_H) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the accumulator holds the value 80 (50_H). Register B holds the value 160 (0A0_H). The instruction

MUL AB

will give the product 12,800 (3200_H), so B is changed to 32_H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

Operation: MUL

$$\begin{matrix} (A7-0) \\ (B15-8) \end{matrix} \leftarrow (A) \times (B)$$

Encoding:

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 4

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
```

Operation: NOP

Encoding:

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 0C3_H (11000011B) and R0 holds 55_H (01010101B) then the instruction

ORL A,R0

will leave the accumulator holding the value 0D7_H (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**

Operation: ORL
 $(A) \leftarrow (A) \vee (Rn)$

Encoding:

0 1 0 0	1 r r r
---------	---------

Bytes: 1

Cycles: 1

ORL A,direct

Operation: ORL
 $(A) \leftarrow (A) \vee (\text{direct})$

Encoding:

0 1 0 0	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

ORL A,@Ri

Operation: ORL
 $(A) \leftarrow (A) \vee ((Ri))$

Encoding:

0 1 0 0	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

ORL A,#data

Operation: ORL
 $(A) \leftarrow (A) \vee \#data$

Encoding:

0 1 0 0	0 1 0 0
---------	---------

immediate data

Bytes: 2

Cycles: 1

ORL direct,A

Operation: ORL
 $(\text{direct}) \leftarrow (\text{direct}) \vee (A)$

Encoding:

0 1 0 0	0 0 1 0
---------	---------

direct address

Bytes: 2

Cycles: 1

ORL **direct, #data**

Operation: ORL
 (direct) ← (direct) ∨ #data

Encoding:

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

direct address

immediate data

Bytes: 3

Cycles: 2

ORL C, <src-bit>

Function: Logical OR for bit variables

Description: Set the carry flag if the Boolean value is a logic 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 P1.0
ORL    C,ACC.7         ; OR carry with the accumulator bit 7
ORL    C,/OV           ; OR carry with the inverse of OV
```

ORL C,bit

Operation: ORL
 $(C) \leftarrow (C) \vee (\text{bit})$

Encoding:

0	1	1	1	0	0	1	0
---	---	---	---	---	---	---	---

bit address

Bytes: 2

Cycles: 2

ORL C,/bit

Operation: ORL
 $(C) \leftarrow (C) \vee \neg (\text{bit})$

Encoding:

1	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

bit address

Bytes: 2

Cycles: 2

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 the transfer to the directly addressed byte indicated. No flags are affected.

Example: The stack pointer originally contains the value 32_H, and internal RAM locations 30_H through 32_H contain the values 20_H, 23_H, and 01_H, respectively. The instruction sequence

```
POP    DPH
POP    DPL
```

will leave the stack pointer equal to the value 30_H and the data pointer set to 0123_H. At this point the instruction

```
POP    SP
```

will leave the stack pointer set to 20_H. Note that in this special case the stack pointer was decremented to 2F_H before being loaded with the value popped (20_H).

Operation: POP
 (direct) ← ((SP))
 (SP) ← (SP) – 1

Encoding:

1 1 0 1	0 0 0 0
---------	---------

direct address

Bytes: 2

Cycles: 2

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 09_H. The data pointer holds the value 0123_H. The instruction sequence

```
PUSH   DPL
PUSH   DPH
```

will leave the stack pointer set to 0B_H and store 23_H and 01_H in internal RAM locations 0A_H and 0B_H, respectively.

Operation: PUSH
 (SP) ← (SP) + 1
 ((SP)) ← (direct)

Encoding:

1 1 0 0	0 0 0 0
---------	---------

direct address

Bytes: 2

Cycles: 2

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 0B_H. Internal RAM locations 0A_H and 0B_H contain the values 23_H and 01_H, respectively. The instruction

RET

will leave the stack pointer equal to the value 09_H. Program execution will continue at location 0123_H.

Operation: RET
 $(PC_{15-8}) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) - 1$
 $(PC_{7-0}) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) - 1$

Encoding:

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 2

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 is 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 0B_H. An interrupt was detected during the instruction ending at location 0122_H. Internal RAM locations 0A_H and 0B_H contain the values 23_H and 01_H, respectively. The instruction
 RETI
 will leave the stack pointer equal to 09_H and return program execution to location 0123_H.

Operation: RETI
 (PC15-8) ← ((SP))
 (SP) ← (SP) – 1
 (PC7-0) ← ((SP))
 (SP) ← (SP) – 1

Encoding:

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 2

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 0C5_H (11000101B). The instruction

RL A

leaves the accumulator holding the value 8B_H (10001011B) with the carry unaffected.

Operation: RL
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$
 $(A_0) \leftarrow (A_7)$

Encoding:

0 0 1 0	0 0 1 1
---------	---------

Bytes: 1

Cycles: 1

RLC A

Function: Rotate accumulator left through 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 0C5_H (11000101B), and the carry is zero. The instruction

RLC A

leaves the accumulator holding the value 8A_H (10001010B) with the carry set.

Operation: RLC
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$
 $(A_0) \leftarrow (C)$
 $(C) \leftarrow (A_7)$

Encoding:

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 0C5_H (11000101B). The instruction

RR A

leaves the accumulator holding the value 0E2_H (11100010B) with the carry unaffected.

Operation: RR

$(A_n) \leftarrow (A_{n+1}) \quad n = 0-6$

$(A_7) \leftarrow (A_0)$

Encoding:

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 0C5_H (11000101B), the carry is zero. The instruction

```
RRC A
```

leaves the accumulator holding the value 62_H (01100010B) with the carry set.

Operation: RRC
 $(A_n) \leftarrow (A_{n+1}) \quad n=0-6$
 $(A_7) \leftarrow (C)$
 $(C) \leftarrow (A_0)$

Encoding:

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 34_H (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 35_H (00110101B).

SETB C

Operation: SETB
(C) ← 1

Encoding:

1	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

SETB bit

Operation: SETB
(bit) ← 1

Encoding:

1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

bit address

Bytes: 2

Cycles: 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 0123_H. The instruction
SJMP RELADR
 will assemble into location 0100_H. After the instruction is executed, the PC will contain the value 0123_H.

Note:

Under the above conditions the instruction following SJMP will be at 102_H. Therefore, the displacement byte of the instruction will be the relative offset (0123_H - 0102_H) = 21_H. In other words, an SJMP with a displacement of 0FE_H would be a one-instruction infinite loop.

Operation: **SJMP**
 $(PC) \leftarrow (PC) + 2$
 $(PC) \leftarrow (PC) + rel$

Encoding:

1 0 0 0	0 0 0 0
---------	---------

rel. address

Bytes: 2

Cycles: 2

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 0C9_H (11001001B), register 2 holds 54_H (01010100B), and the carry flag is set. The instruction

```
SUBB A,R2
```

will leave the value 74_H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9_H minus 54_H is 75_H. The difference between this and the above result is due to the (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 be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Operation: SUBB
 $(A) \leftarrow (A) - (C) - (Rn)$

Bytes: 1

Cycles: 1

SUBB A, direct

Operation: SUBB
 $(A) \leftarrow (A) - (C) - (\text{direct})$

Encoding:

1 0 0 1	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

SUBB A, @ Ri

Operation: SUBB
 $(A) \leftarrow (A) - (C) - ((Ri))$

Encoding:

1 0 0 1	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

SUBB A, #data

Operation: SUBB
 $(A) \leftarrow (A) - (C) - \#data$

Encoding:

1 0 0 1	0 1 0 0
---------	---------

immediate data

Bytes: 2

Cycles: 1

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 0C5_H (11000101B). The instruction

SWAP A

leaves the accumulator holding the value 5C_H (01011100B).

Operation: SWAP
(A3-0) \leftrightarrow (A7-4), (A7-4) \leftarrow (A3-0)

Encoding:

1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 20_H. The accumulator holds the value 3F_H (00111111B). Internal RAM location 20_H holds the value 75_H (01110101B). The instruction

XCH A, @R0

will leave RAM location 20_H holding the value 3F_H (00111111 B) and 75_H (01110101B) in the accumulator.

XCH A,Rn

Operation: XCH
 (A) ⇌ (Rn)

Encoding:

1	1	0	0
---	---	---	---

1	r	r	r
---	---	---	---

Bytes: 1

Cycles: 1

XCH A,direct

Operation: XCH
 (A) ⇌ (direct)

Encoding:

1	1	0	0
---	---	---	---

0	1	0	1
---	---	---	---

direct address

Bytes: 2

Cycles: 1

XCH A, @ Ri

Operation: XCH
 (A) \leftrightarrow ((Ri))

Encoding:

1	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 20_H. The accumulator holds the value 36_H (00110110B). Internal RAM location 20_H holds the value 75_H (01110101B). The instruction
 XCHD A, @ R0
 will leave RAM location 20_H holding the value 76_H (01110110B) and 35_H (00110101B) in the accumulator.

Operation: XCHD
 (A3-0) \leftrightarrow ((Ri)3-0)

Encoding:

1	1	0	1	0	1	i
---	---	---	---	---	---	---

Bytes: 1

Cycles: 1

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 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 0C3_H (11000011B) and register 0 holds 0AA_H (10101010B) then the instruction

XRL A,R0

will leave the accumulator holding the value 69_H (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

Operation: XRL2
 (A) ← (A) ∨ (Rn)

Encoding:

0 1 1 0	1 r r r
---------	---------

Bytes: 1

Cycles: 1

XRL A, direct

Operation: XRL
 $(A) \leftarrow (A) \vee (\text{direct})$

Encoding:

0 1 1 0	0 1 0 1
---------	---------

direct address

Bytes: 2

Cycles: 1

XRL A, @ Ri

Operation: XRL
 $(A) \leftarrow (A) \vee ((Ri))$

Encoding:

0 1 1 0	0 1 1 i
---------	---------

Bytes: 1

Cycles: 1

XRL A, #data

Operation: XRL
 $(A) \leftarrow (A) \vee \#data$

Encoding:

0 1 1 0	0 1 0 0
---------	---------

immediate data

Bytes: 2

Cycles: 1

XRL direct, A

Operation: XRL
 $(\text{direct}) \leftarrow (\text{direct}) \vee (A)$

Encoding:

0 1 1 0	0 0 1 0
---------	---------

direct address

Bytes: 2

Cycles: 1

XRL **direct, #data**

Operation: XRL
 (direct) ← (direct) ∨ #data

Encoding:

0	1	1	0
0	0	1	1

direct address

immediate data

Bytes: 3

Cycles: 2

Instruction Set Summary

Mnemonic	Description	Byte	Cycle
Arithmetic Operations			
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 A with carry flag	2	1
ADDC A,@Ri	Add indirect RAM to A with carry flag	1	1
ADDC A,#data	Add immediate data to A with carry flag	2	1
SUBB A,Rn	Subtract register from A with borrow	1	1
SUBB A,direct	Subtract direct byte from A with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB A,#data	Subtract immediate data from A 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 A and B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal adjust accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Logic Operations			
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
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 byte	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	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within the accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Data Transfer			
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	Move direct byte to indirect RAM	2	2
MOV @Ri,#data	Move immediate data to indirect RAM	2	1
MOV DPTR,#data16	Load data pointer with a 16-bit constant	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 addr.) to A	1	2
MOVX A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX @Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from 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 low-order nibble indir. RAM with A	1	1

*) MOV A,ACC is not a valid instruction

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
----------	-------------	------	-------

Boolean Variable Manipulation

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

Program and Machine Control

ACALL addr11	Absolute subroutine call	2	2
LCALL addr16	Long subroutine call	3	2
RET	Return from subroutine	1	2
RETI	Return from interrupt	1	2
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 flag is not set	2	2
JB bit,rel	Jump if direct bit is set	3	2
JNB bit,rel	Jump if direct bit is not set	3	2
JBC bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE A,direct,rel	Compare direct byte to A and jump if not equal	3	2

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
----------	-------------	------	-------

Program and Machine Control (cont'd)

CJNE A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE @Ri,#data,rel	Compare immed. to ind. 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
NOP	No operation	1	1