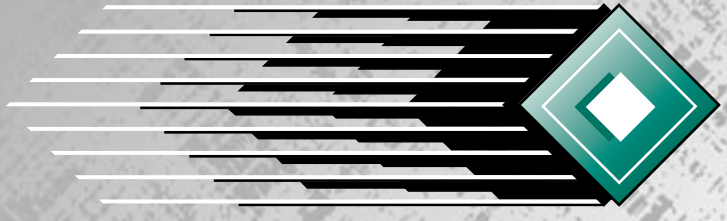
8XC196NP, 80C196NU Microcontroller User's Manual



intel®

8XC196NP, 80C196NU Microcontroller User's Manual

August 2004 **Order Number 272479-003**

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

CONTENTS

CHAPTER 3
ADVANCED MATH FEATURES

CHAPTER 4
PROGRAMMING CONSIDERATIONS

CHAPTER 6
STANDARD AND PTS INTERRUPTS

CHAPTER 7
I/O PORTS

FIGURES

FIGURES

FIGURES

TABLES

TABLES

TABLES

1

Guide to This Manual

CHAPTER 1
GUIDE TO THIS MANUAL

This manual describes the 8XC196NP and 80C196NU embedded microcontrollers. It is intended for use by both software and hardware designers familiar with the principles of microcontrollers. This chapter describes what you'll find in this manual, lists other documents that may be useful, and explains how to access the support services we provide to help you complete your design.

1.1 MANUAL CONTENTS

This manual contains several chapters and appendixes, a glossary, and an index. This chapter, Chapter 1, provides an overview of the manual. This section summarizes the contents of the remaining chapters and appendixes. The remainder of this chapter describes notational conventions and terminology used throughout the manual, provides references to related documentation, describes customer support services, and explains how to access information and assistance.

Chapter 2 — Architectural Overview — provides an overview of the device hardware. It describes the core, internal timing, internal peripherals, and special operating modes.

Chapter 3 — Advanced Math Features — describes the advanced mathematical features of the 80C196NU. The 80C196NU is the first member of the MCS® 96 microcontroller family to incorporate enhanced 16-bit multiplication instructions for performing multiply-accumulate operations and a dedicated, 32-bit accumulator register for storing the results of these operations. The accumulator and the enhanced instructions combine to decrease the amount of time required to perform multiply-accumulate operations. The instructions and accumulator support signed and unsigned integers as well as signed fractional data.

Chapter 4 — Programming Considerations — provides an overview of the instruction set, describes general standards and conventions, and defines the operand types and addressing modes supported by the MCS® 96 microcontroller family. (For additional information about the instruction set, see Appendix A.)

Chapter 5 — Memory Partitions — describes the addressable memory space of the device. It describes the memory partitions, explains how to use windows to increase the amount of memory that can be accessed with direct addressing, and provides examples of memory configurations.

Chapter 6 — Standard and PTS Interrupts — describes the interrupt control circuitry, priority scheme, and timing for standard and peripheral transaction server (PTS) interrupts. It also explains interrupt programming and control.

Chapter 7 — I/O Ports — describes the input/output ports and explains how to configure the ports for input, output, or special functions.

Chapter 8 — Serial I/O (SIO) Port — describes the asynchronous/synchronous serial I/O (SIO) port and explains how to program it.

Chapter 9 —Pulse-width Modulator — provides a functional overview of the pulse width modulator (PWM) modules, describes how to program them, and provides sample circuitry for converting the PWM outputs to analog signals.

Chapter 10 — Event Processor Array (EPA) — describes the event processor array, a timer/counter-based, high-speed input/output unit. It describes the timer/counters and explains how to program the EPA and how to use the EPA to produce pulse-width modulated (PWM) outputs.

Chapter 11 — Minimum Hardware Considerations — describes options for providing the basic requirements for device operation within a system, discusses other hardware considerations, and describes device reset options.

Chapter 12 — Special Operating Modes — provides an overview of the idle, powerdown, standby, and on-circuit emulation (ONCE) modes and describes how to enter and exit each mode.

Chapter 13 — Interfacing with External Memory — lists the external memory signals and describes the registers that control the external memory interface. It discusses the chip selects, multiplexed and demultiplexed bus modes, bus width and memory configurations, the bus-hold protocol, write-control modes, and internal wait states and ready control. Finally, it provides timing information for the system bus.

Appendix A — Instruction Set Reference — provides reference information for the instruction set. It describes each instruction; defines the processor status word (PSW) flags; shows the relationships between instructions and PSW flags; and lists hexadecimal opcodes, instruction lengths, and execution times. (For additional information about the instruction set, see Chapter 4, "Programming Considerations.")

Appendix B — Signal Descriptions — provides reference information for the device pins, including descriptions of the pin functions, reset status of the I/O and control pins, and package pin assignments.

Appendix C — Registers — provides a compilation of all device special-function registers (SFRs) arranged alphabetically by register mnemonic. It also includes tables that list the windowed direct addresses for all SFRs in each possible window.

Glossary — defines terms with special meaning used throughout this manual.

Index — lists key topics with page number references.

1.2 NOTATIONAL CONVENTIONS AND TERMINOLOGY

The following notations and terminology are used throughout this manual. The Glossary defines other terms with special meanings.

— The pound symbol (#) has either of two meanings, depending on the context. When used with a signal name, the symbol means that the signal is active low. When used in an instruction, the symbol prefixes an immediate value in immediate addressing mode.

addresses — In this manual, both internal and external addresses use the number of hexadecimal digits that correspond with the number of available address lines. For example, the highest possible internal address is shown as FFFFFFH, while the highest possible external address is shown as FFFFFH. When writing code, use the appropriate address conventions for the software tool you are using. (In general, assemblers require a zero preceding an alphabetic hexadecimal character and an "H" following any hexadecimal value, so FFFFFFH must be written as 0FFFFFFH. ANSI 'C' compilers require a zero plus an "x" preceding a hexadecimal value, so FFFFFFH must be written as 0xFFFFFF.) Consult the manual for your assembler or compiler to determine its specific requirements.

assert and deassert — The terms *assert* and *deassert* refer to the act of making a signal active (enabled) and inactive (disabled), respectively. The active polarity (low or high) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low; to assert ALE is to drive it high; to deassert RD# is to drive it high; to deassert ALE is to drive it low.

clear and set — The terms *clear* and *set* refer to the value of a bit or the act of giving it a value. If a bit is clear, its value is "0"; clearing a bit gives it a "0" value. If a bit is set, its value is "1"; setting a bit gives it a "1" value.

f — Lowercase "f" represents the internal operating frequency. See "Internal Timing" on page 2-7 for details.

instructions — Instruction mnemonics are shown in upper case to avoid confusion. In general, you may use either upper case or lower case when programming. Consult the manual for your assembler or compiler to determine its specific requirements.

italics — Italics identify variables and introduce new terminology. The context in which italics are used distinguishes between the two possible meanings.

Variables in registers and signal names are commonly represented by *x* and *y*, where *x* represents the first variable and *y* represents the second variable. For example, in register P*x*_MODE.*y*, *x* represents the variable that identifies the specific port associated with the register, and *y* represents the register bit variable (7:0 or 15:0). Variables must be replaced with the correct values when configuring or programming registers or identifying signals.

numbers — Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character *H*. Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter *B* is appended to binary numbers for clarity.)

register bits — Bit locations are indexed by 7:0 (or 15:0), where bit 0 is the least-significant bit and bit 7 (or 15) is the most-significant bit. An individual bit is represented by the register name, followed by a period and the bit number. For example, WSR.7 is bit 7 of the window selection register. In some discussions, bit names are used.

register names — Register mnemonics are shown in upper case. For example, TIMER2 is the timer 2 register; timer 2 is the timer. A register name containing a lowercase italic character represents more than one register. For example, the *x* in P*x*_REG indicates that the register name refers to any of the port data registers.

reserved bits — Certain bits are described as *reserved* bits. In illustrations, reserved bits are indicated with a dash (—). These bits are not used in this device, but they may be used in future implementations. To help ensure that a current software design is compatible with future implementations, reserved bits should be cleared (given a value of "0") or left in their default states, unless otherwise noted. Do not rely on the values of reserved bits; consider them undefined.

signal names — Signal names are shown in upper case. When several signals share a common name, an individual signal is represented by the signal name followed by a number. For example, the EPA signals are named EPA0, EPA1, EPA2, etc. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P1.0, P1.1); a range of pins is represented by P*x*.*y*:*z* (e.g., P1.4:0 represents five port pins: P1.4, P1.3, P1.2, P1.1, P1.0). A pound symbol (#) appended to a signal name identifies an active-low signal.

t Lowercase “t” represents the internal operating period. See “Internal Timing” on page 2-7 for details.

units of measure The following abbreviations are used to represent units of measure:

A	amps, amperes
DCV	direct current volts
Kbytes	kilobytes
kHz	kilohertz
kΩ	kilo-ohms
mA	milliamps, milliamperes
Mbytes	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
W	watts
V	volts
µA	microamps, microamperes
µF	microfarads
µs	microseconds
µW	microwatts

X Uppercase X (no italics) represents an unknown value or an irrelevant (“don’t care”) state or condition. The value may be either binary or hexadecimal, depending on the context. For example, 2XAFH (hex) indicates that bits 11:8 are unknown; 10XXB (binary) indicates that the two least-significant bits are unknown.

1.3 RELATED DOCUMENTS

The tables in this section list additional documents that you may find useful in designing systems incorporating MCS 96 microcontrollers. These are not comprehensive lists, but are a representative sample of relevant documents. For a complete list of available printed documents, please order the literature catalog (order number 210621). To order documents, please call the Intel literature center for your area (telephone numbers are listed on page 1-11).

Intel’s *Ap*BUILDER software, hypertext manuals and datasheets, and electronic versions of application notes and code examples are also available from the BBS (see “Bulletin Board System (BBS)” on page 1-9). New information is available first from FaxBack and the BBS. Refer to “Electronic Support Systems” on page 1-8 for details.

Table 1-1. Handbooks and Product Information

Title and Description	Order Number
Intel Embedded Quick Reference Guide	272439
Solutions for Embedded Applications Guide	240691
Data on Demand fact sheet	240952
Data on Demand annual subscription (6 issues; Windows* version) Complete set of Intel handbooks on CD-ROM.	240897
Handbook Set — handbooks and product overview Complete set of Intel's product line handbooks. Contains datasheets, application notes, article reprints and other design information on microprocessors, peripherals, embedded controllers, memory components, single-board computers, microcommunications, software development tools, and operating systems.	231003
Automotive Products † Application notes and article reprints on topics including the MCS 51 and MCS 96 microcontrollers. Documents in this handbook discuss hardware and software implementations and present helpful design techniques.	231792
Embedded Applications handbook (2 volume set) † Datasheets, architecture descriptions, and application notes on topics including flash memory devices, networking chips, and MCS 51 and MCS 96 microcontrollers. Documents in this handbook discuss hardware and software implementations and present helpful design techniques.	270648
Embedded Microcontrollers † Datasheets and architecture descriptions for Intel's three industry-standard microcontrollers, the MCS 48, MCS 51, and MCS 96 microcontrollers.	270646
Peripheral Components † Comprehensive information on Intel's peripheral components, including datasheets, application notes, and technical briefs.	296467
Flash Memory (2 volume set) † A collection of datasheets and application notes devoted to techniques and information to help design semiconductor memory into an application or system.	210830
Packaging † Detailed information on the manufacturing, applications, and attributes of a variety of semiconductor packages.	240800
Development Tools Handbook Information on third-party hardware and software tools that support Intel's embedded microcontrollers.	272326

† Included in handbook set (order number 231003)

Table 1-2. Application Notes, Application Briefs, and Article Reprints

Title	Order Number
AB-71, *Using the SIO on the 8XC196MH* (application brief)	272594
AP-125, *Design Microcontroller Systems for Electrically Noisy Environments* †††	210313
AP-155, *Oscillators for Microcontrollers* †††	230659
AR-375, *Motor Controllers Take the Single-Chip Route* (article reprint)	270056
AP-406, *MCS® 96 Analog Acquisition Primer* †††	270365

† Included in *Automotive Products* handbook (order number 231792)
†† Included in *Embedded Applications* handbook (order number 270648)
††† Included in *Automotive Products* and *Embedded Applications* handbooks

Table 1-2. Application Notes, Application Briefs, and Article Reprints (Continued)

Title	Order Number
AP-445, *8XC196KR Peripherals: A User's Point of View* †	270873
AP-449, *A Comparison of the Event Processor Array (EPA) and High Speed Input/Output (HSIO) Unit* †	270968
AP-475, *Using the 8XC196NT* ††	272315
AP-477, *Low Voltage Embedded Design* ††	272324
AP-483, *Application Examples Using the 8XC196MC/MD Microcontroller*	272282
AP-700, *Intel Fuzzy Logic Tool Simplifies ABS Design* †	272595
AP-711, *EMI Design Techniques for Microcontrollers in Automotive Applications*	272324
AP-715, *Interfacing an I^2C Serial EEPROM to an MCS® 96 Microcontroller*	272680

† Included in *Automotive Products* handbook (order number 231792)
†† Included in *Embedded Applications* handbook (order number 270648)
††† Included in *Automotive Products* and *Embedded Applications* handbooks

Table 1-3. MCS® 96 Microcontroller Datasheets (Commercial/Express)

Title	Order Number
8XC196KR/KQ/JR/JQ Commercial/Express CHMOS Microcontroller †	270912
8XC196KT Commercial CHMOS Microcontroller †	272266
87C196KT/87C196KS 20 MHz Advanced 16-Bit CHMOS Microcontroller †	272513
8XC196MC Industrial Motor Control Microcontroller †	272323
87C196MD Industrial Motor Control CHMOS Microcontroller †	270946
8XC196NP Commercial CHMOS 16-Bit Microcontroller †	272459
8XC196NT CHMOS Microcontroller with 1-Mbyte Linear Address Space †	272267
80C196NU Commercial CHMOS 16-Bit Microcontroller	272644

† Included in *Embedded Microcontrollers* handbook (order number 270646)

Table 1-4. MCS® 96 Microcontroller Datasheets (Automotive)

Title and Description	Order Number
87C196CA/87C196CB 20 MHz Advanced 16-Bit CHMOS Microcontroller with Integrated CAN 2.0 †	272405
87C196JT 20 MHz Advanced 16-Bit CHMOS Microcontroller †	272529
87C196JV 20 MHz Advanced 16-Bit CHMOS Microcontroller †	272580
87C196KR/KQ, 87C196JV/JT, 87C196JR/JQ Advanced 16-Bit CHMOS Microcontroller †	270827
87C196KT/87C196KS Advanced 16-Bit CHMOS Microcontroller †	270999
87C196KT/KS 20 MHz Advanced 16-Bit CHMOS Microcontroller †	272513

† Included in *Automotive Products* handbook (order number 231792)

Table 1-5. MCS® 96 Microcontroller Quick References

Title and Description	Order Number
8XC196KR Quick Reference (includes the JQ, JR, KQ, KR)	272113
8XC196KT Quick Reference	272269
8XC196MC Quick Reference	272114
8XC196NP Quick Reference	272466
8XC196NT Quick Reference	272270

Page Intentionally Left Blank

Page Intentionally Left Blank

1.4.4 World Wide Web

We offer a variety of information through the World Wide Web (URL:http://www.intel.com/). Select “Embedded Design Products” from the Intel home page.

1.5 TECHNICAL SUPPORT

In the U.S. and Canada, technical support representatives are available to answer your questions between 5 a.m. and 5 p.m. PST. You can also fax your questions to us. (Please include your voice telephone number and indicate whether you prefer a response by phone or by fax). Outside the U.S. and Canada, please contact your local distributor.

1-800-628-8686	U.S. and Canada
916-356-7599	U.S. and Canada
916-356-6100 (fax)	U.S. and Canada

1.6 PRODUCT LITERATURE

You can order product literature from the following Intel literature centers.

1-800-468-8118, ext. 283	U.S. and Canada
708-296-9333	U.S. (from overseas)
44(0)1793-431155	Europe (U.K.)
44(0)1793-421333	Germany
44(0)1793-421777	France
81(0)120-47-88-32	Japan (fax only)

2

Architectural Overview

CHAPTER 2
ARCHITECTURAL OVERVIEW

The 16-bit 8XC196NP and 80C196NU CHMOS microcontrollers are designed to handle high-speed calculations and fast input/output (I/O) operations. They share a common architecture and instruction set with other members of the MCS® 96 microcontroller family. In addition to their 16-bit address/data buses, both microcontrollers have extended addressing ports consisting of 4 external address pins, for a total of 20 address pins. With 20 address pins, these microcontrollers can access up to 1 Mbyte of linear address space. Both devices also have chip-select units that provide a glueless interface to external memory devices. The extended addressing port and chip-select unit enable these microcontrollers to handle larger, more complex programs and to access more external memory at a faster rate than could earlier MCS 96 microcontrollers.

The 8XC196NP and 80C196NU are pin-compatible and have identical cores. However, the 80C196NU can operate at twice the frequency of the 8XC196NP. The 80C196NU also employs an accumulator and enhanced multiplication instructions to support multiply-accumulate operations. The 80C196NU is the first MCS 96 microcontroller with this capability. This chapter provides a high-level overview of the architecture.

2.1 TYPICAL APPLICATIONS

MCS 96 microcontrollers are typically used for high-speed event control systems. Commercial applications include modems, motor-control systems, printers, photocopiers, air conditioner control systems, disk drives, and medical instruments. Automotive customers use MCS 96 microcontrollers in engine-control systems, airbags, suspension systems, and antilock braking systems (ABS).

2.2 DEVICE FEATURES

Table 2-1 lists the features of the 8XC196NP and 80C196NU.

Table 2-1. Features of the 8XC196NP and 80C196NU

Device	Pins	ROM (Note 1)	Register RAM (Note 2)	I/O Pins (Note 3)	EPA Pins	SIO Ports	PWM Channels	Chip-select Pins	External Interrupt Pins
8XC196NP	100	4 K	1024	64	4	1	3	6	4
80C196NU	100	0	1024	64	4	1	3	6	4

NOTES:
1. Nonvolatile memory is optional for the 8XC196NP, but is not available for the 80C196NU. The second character of the device name indicates the presence and type of nonvolatile memory. 80C196NP = none; 83C196NP = ROM.
2. Register RAM amounts include the 24 bytes allocated to core special-function registers (SFRs) and the stack pointer.
3. I/O pins include address, data, and bus control pins and 32 I/O port pins.

2.3 BLOCK DIAGRAM

Figure 2-1 shows the major blocks within the device. The core of the device (Figure 2-2) consists of the central processing unit (CPU) and memory controller. The CPU contains the register file and the register arithmetic-logic unit (RALU). The CPU connects to both the memory controller and an interrupt controller via a 16-bit internal bus. An extension of this bus connects the CPU to the internal peripheral modules. In addition, an 8-bit internal bus transfers instruction bytes from the memory controller to the instruction register in the RALU.

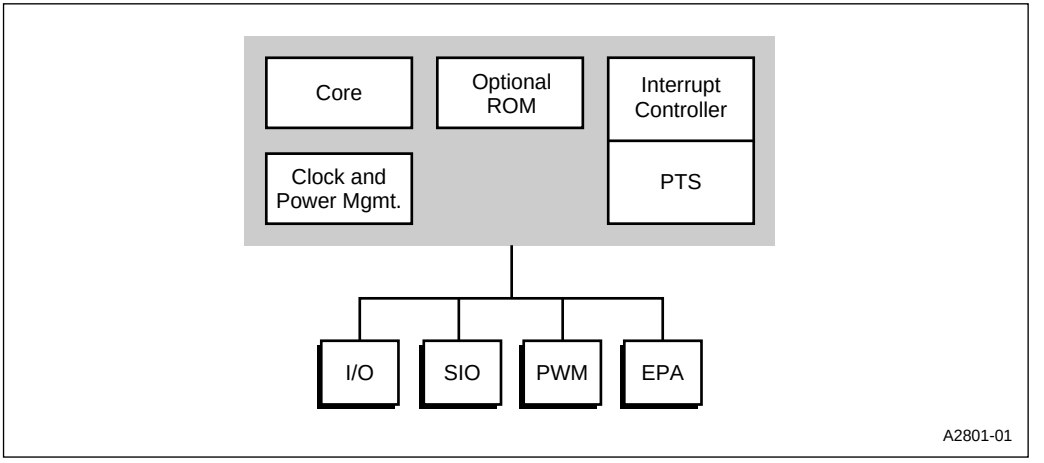


Figure 2-1. 8XC196NP and 80C196NU Block Diagram

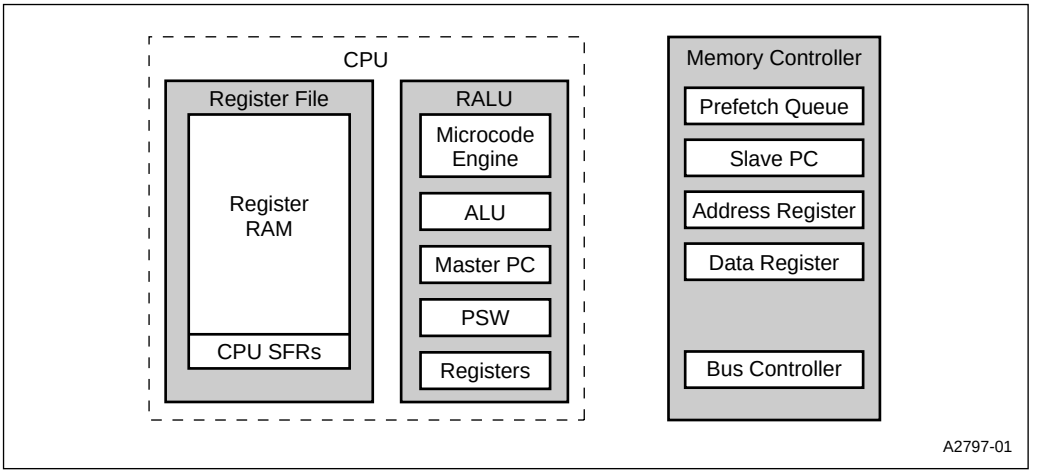


Figure 2-2. Block Diagram of the Core

2.3.1 CPU Control

The CPU is controlled by the microcode engine, which instructs the RALU to perform operations using bytes, words, or double words from either the 256-byte lower register file or through a *window* that directly accesses the upper register file. (See Chapter 5, “Memory Partitions,” for more information about the register file and windowing.) CPU instructions move from the 4-byte (for the 8XC196NP) or 8-byte (for the 80C196NU) prefetch queue in the memory controller into the RALU’s instruction register. The microcode engine decodes the instructions and then generates the sequence of events that cause desired functions to occur.

2.3.2 Register File

The register file is divided into an upper and a lower file. In the lower register file, the lowest 24 bytes are allocated to the CPU’s special-function registers (SFRs) and the stack pointer, while the remainder is available as general-purpose register RAM. The upper register file contains only general-purpose register RAM. The register RAM can be accessed as bytes, words, or double-words.

The RALU accesses the upper and lower register files differently. The lower register file is always directly accessible with direct addressing (see “Addressing Modes” on page 4-6). The upper register file is accessible with direct addressing only when *windowing* is enabled. Windowing is a technique that maps blocks of the upper register file into a *window* in the lower register file. See Chapter 5, “Memory Partitions,” for more information about the register file and windowing.

2.3.3 Register Arithmetic-logic Unit (RALU)

The RALU contains the microcode engine, the 16-bit arithmetic logic unit (ALU), the master program counter (PC), the processor status word (PSW), and several registers. The registers in the RALU are the instruction register, a constants register, a bit-select register, a loop counter, and three temporary registers (the upper-word, lower-word, and second-operand registers).

The 24-bit master program counter (PC) provides a linear, nonsegmented 16-Mbyte memory space. Only 20 of the address lines are implemented with external pins, so you can physically address only 1 Mbyte. (For compatibility with earlier devices, the PC can be configured as 16 bits wide.) The master PC contains the address of the next instruction and has a built-in incrementer that automatically loads the next sequential address. However, if a jump, interrupt, call, or return changes the address sequence, the ALU loads the appropriate address into the master PC.

The PSW contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of your program. Appendix A, "Instruction Set Reference," provides a detailed description of the PSW.

All registers, except the 3-bit bit-select register and the 6-bit loop counter, are either 16 or 17 bits (16 bits plus a sign extension). Some of these registers can reduce the ALU's workload by performing simple operations.

The RALU uses the upper- and lower-word registers together for the 32-bit instructions and as temporary registers for many instructions. These registers have their own shift logic and are used for operations that require logical shifts, including normalize, multiply, and divide operations. The six-bit loop counter counts repetitive shifts. The second-operand register stores the second operand for two-operand instructions, including the multiplier during multiply operations and the divisor during divide operations. During subtraction operations, the output of this register is complemented before it is moved into the ALU.

The RALU speeds up calculations by storing constants (e.g., 0, 1, and 2) in the constants register so that they are readily available when complementing, incrementing, or decrementing bytes or words. In addition, the constants register generates single-bit masks, based on the bit-select register, for bit-test instructions.

2.3.3.1 Code Execution

The RALU performs most calculations for the device, but it does not use an *accumulator*. Instead it operates directly on the lower register file, which essentially provides 256 accumulators. Because data does not flow through a single accumulator, the device's code executes faster and more efficiently.

2.3.3.2 Instruction Format

MCS 96 microcontrollers combine a large set of general-purpose registers with a three-operand instruction format. This format allows a single instruction to specify two source registers and a separate destination register. For example, the following instruction multiplies two 16-bit variables and stores the 32-bit result in a third variable.

```
MUL  RESULT, FACTOR_1, FACTOR_2     ;multiply FACTOR_1 and FACTOR_2
                                    ;and store answer in RESULT
                                    ;(RESULT)←(FACTOR_1 × FACTOR_2)
```

An 80C186 device requires four instructions to accomplish the same operation. The following example shows the equivalent code for an 80C186 device.

```
MOV  AX, FACTOR_1                   ;move FACTOR_1 into accumulator (AX)
                                    ;(AX)←FACTOR1
MUL  FACTOR_2                       ;multiply FACTOR_2 and AX
                                    ;(DX:AX)←(AX)×(FACTOR_2)
MOV  RESULT, AX                     ;move lower byte into RESULT
                                    ;(RESULT)←(AX)
MOV  RESULT+2, DX                   ;move upper byte into RESULT+2
                                    ;(RESULT+2)←(DX)
```

2.3.4 Memory Controller

The RALU communicates with all memory, except the register file and peripheral SFRs, through the memory controller. (It communicates with the upper register file through the memory controller except when *windowing* is used; see Chapter 5, “Memory Partitions,”) The memory controller contains the prefetch queue, the slave program counter (slave PC), address and data registers, and the bus controller.

The bus controller drives the memory bus, which consists of an internal memory bus and the external address/data bus. The bus controller receives memory-access requests from either the RALU or the prefetch queue; queue requests always have priority. This queue is transparent to the RALU and your software.

NOTE

When using a logic analyzer to debug code, remember that instructions are preloaded into the prefetch queue and are not necessarily executed immediately after they are fetched.

When the bus controller receives a request from the queue, it fetches the code from the address contained in the slave PC. The slave PC increases execution speed because the next instruction byte is available immediately and the processor need not wait for the master PC to send the address to the memory controller. If a jump, interrupt, call, or return changes the address sequence, the master PC loads the new address into the slave PC, then the CPU flushes the queue and continues processing.

The extended program counter (EPC) is an extension of the slave PC. The EPC generates the upper eight address bits for extended code fetches and outputs them on the extended addressing port (EPORT). Because only four EPORT pins are implemented, only the lower four address bits are available. (See Chapter 5, "Memory Partitions," for additional information.)

The memory controller includes a chip-select unit with six chip-select outputs for selecting an external device during an external bus cycle. During an external memory access, a chip-select output is asserted if the address falls within the address range assigned to that chip-select. The bus width, the number of wait states, and multiplexed or demultiplexed address/data lines are programmed independently for the six chip-selects. The address range of the chip-selects can be programmed for various granularities: 256 bytes, 512 bytes, … 512 Kbytes, or 1 Mbyte. The base address can be any address that is evenly divisible by the selected address range. See Chapter 13, "Interfacing with External Memory," for more information.

2.3.5 Multiply-accumulate (80C196NU Only)

The 80C196NU is able to process multiply-accumulate operations through the use of a hardware accumulator and enhanced multiplication instructions. The accumulator includes a 16-bit adder, a 3-to-1 multiplexer, a 32-bit accumulator register, and a control register. The multiply-accumulate function is enabled by any 16-bit multiplication instruction with a destination address that is in the range 00–0FH. The instructions can operate on signed integers, unsigned integers, and signed fractional numbers. The control register allows you to enable *saturation mode* and *fractional mode* for signed multiplication. Chapter 3, "Advanced Math Features," describes the accumulator.

2.3.6 Interrupt Service

The device's flexible interrupt-handling system has two main components: the programmable interrupt controller and the peripheral transaction server (PTS). The programmable interrupt controller has a hardware priority scheme that can be modified by your software. Interrupts that go through the interrupt controller are serviced by interrupt service routines that you provide. The peripheral transaction server (PTS), a microcoded hardware interrupt processor, provides high-speed, low-overhead interrupt handling. You can configure most interrupts (except NMI, trap, and unimplemented opcode) to be serviced by the PTS instead of the interrupt controller.

The PTS can transfer bytes or words, either individually or in blocks, between any memory locations and can generate pulse-width modulated (PWM) signals. PTS interrupts have a higher priority than standard interrupts and may temporarily suspend interrupt service routines. See Chapter 6, "Standard and PTS Interrupts," for more information.

2.4 INTERNAL TIMING

The clock circuitry of the 8XC196NP (Figure 2-3) is identical to that of earlier MCS 96 microcontrollers. It receives an input clock signal on XTAL1 provided by an external crystal or clock and divides the frequency by two. The clock generators accept the divided input frequency from the divide-by-two circuit and produce two nonoverlapping internal timing signals, PH1 and PH2. These signals are active when high.

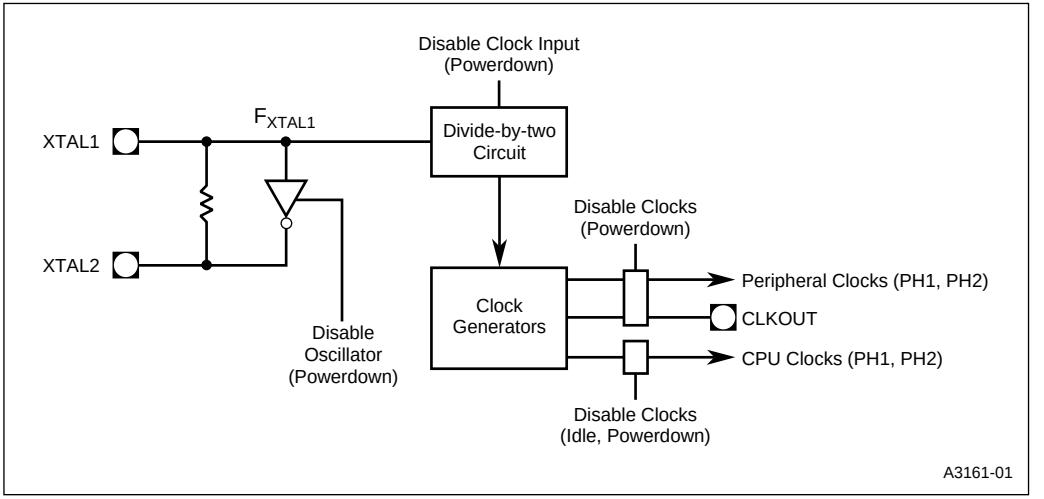


Figure 2-3. Clock Circuitry (8XC196NP)

The 80C196NU's clock circuitry (Figure 2-4) implements phase-locked loop and clock multiplier circuitry, which can substantially increase the CPU clock rate while using a lower-frequency input clock. The clock circuitry accepts an input clock signal on XTAL1 provided by an external crystal or oscillator. Depending on the values of the PLLEN1 and PLLEN2 pins, this frequency is routed either through the phase-locked loop and multiplier or directly to the divide-by-two circuit. The multiplier circuitry can double or quadruple the input frequency (F_{XTAL1}) before the frequency (f) reaches the divide-by-two circuitry. The clock generators accept the divided input frequency (f/2) from the divide-by-two circuit and produce two nonoverlapping internal timing signals, PH1 and PH2. These signals are active when high.

NOTE

For brevity, this manual uses lowercase "f" to represent the internal clock frequency of both the 8XC196NP and the 80C196NU. For the 8XC196NP, f is equal to F_{XTAL1}. For the 80C196NU, f is equal to either F_{XTAL1}, $2F_{XTAL1}$, or $4F_{XTAL1}$, depending on the clock multiplier mode, which is controlled by the PLLEN1 and PLLEN2 input pins.

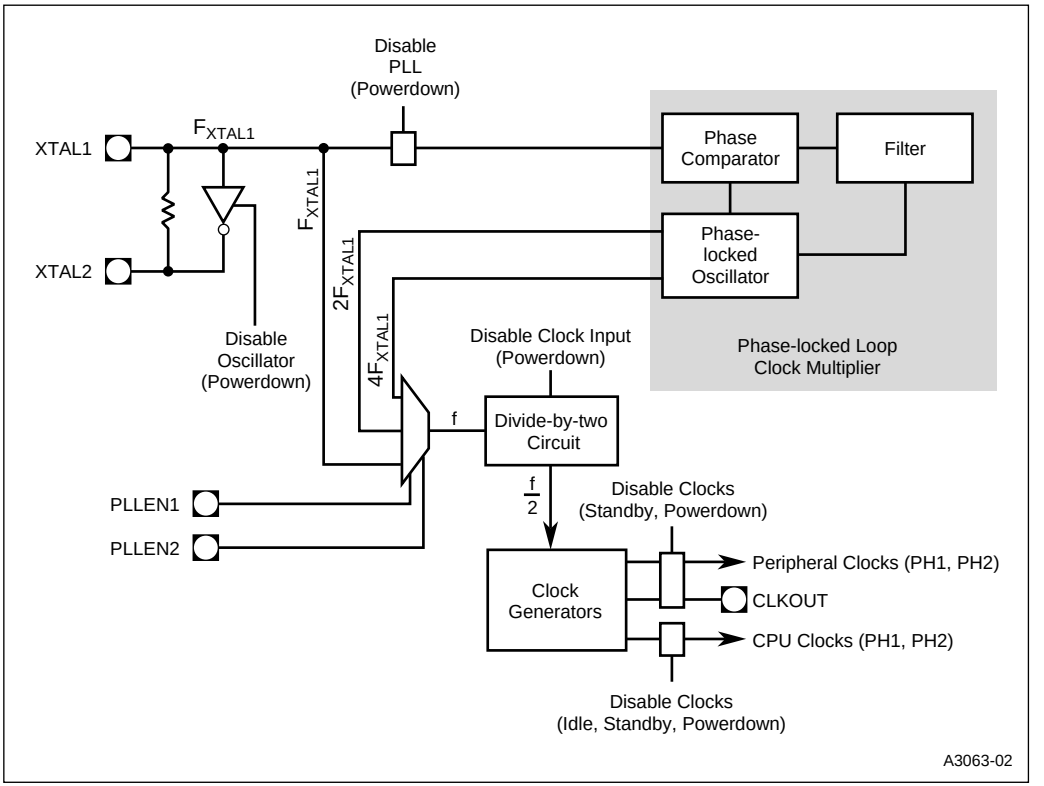


Figure 2-4. Clock Circuitry (80C196NU)

For both the 8XC196NP and 80C196NU, the rising edges of PH1 and PH2 generate CLKOUT (Figure 2-5). The clock circuitry routes separate internal clock signals to the CPU and the peripherals to provide flexibility in power management. ("Reducing Power Consumption" on page 12-3 describes the power management modes.) It also outputs the CLKOUT signal on the CLKOUT pin. Because of the complex logic in the clock circuitry, the signal on the CLKOUT pin is a delayed version of the internal CLKOUT signal. This delay varies with temperature and voltage.

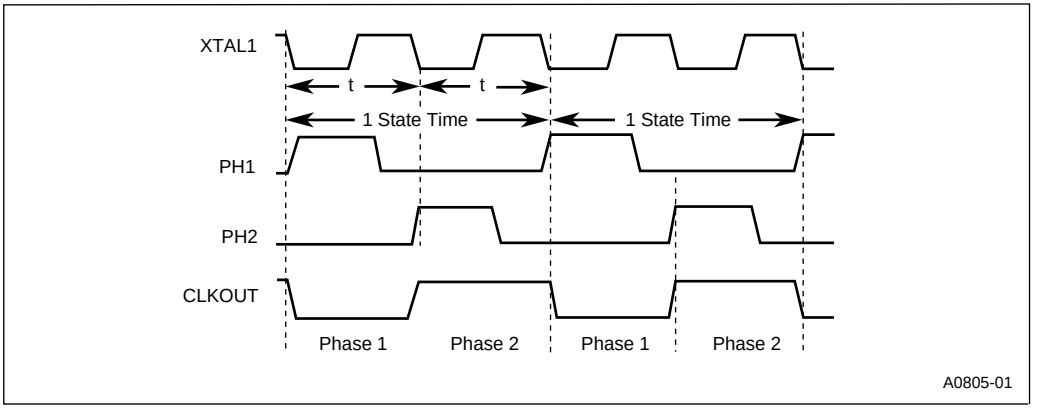


Figure 2-5. Internal Clock Phases

The combined period of phase 1 and phase 2 of the internal CLKOUT signal defines the basic time unit known as a *state time* or *state*. Table 2-2 lists state time durations at various frequencies.

Table 2-2. State Times at Various Frequencies

f (Frequency Input to the Divide-by-two Circuit)	State Time
12.5 MHz	160 ns
25 MHz	80 ns
50 MHz	40 ns

The following formulas calculate the frequency of PH1 and PH2, the duration of a state time, and the duration of a clock period (t).

$$\text{PH1 (in MHz)} = \frac{f}{2} = \text{PH2} \qquad \text{State Time (in µs)} = \frac{2}{f} \qquad t = \frac{1}{f}$$

Because the device can operate at many frequencies, this manual defines time requirements (such as instruction execution times) in terms of state times rather than specific measurements. Datasheets list AC characteristics in terms of clock periods (t).

For the 80C196NU, Table 2-3 details the relationships between the input frequency (F_{XTAL1}), the configuration of PLLEN1 and PLLEN2, the operating frequency (f), the clock period (t), and state times. Figure 2-6 illustrates the timing relationships between the input frequency (F_{XTAL1}), the operating frequency (f), and the CLKOUT signal with each of the three valid PLLEN*x* pin configurations. (Since the maximum operating frequency is 50 MHz, only a 12.5 MHz external clock frequency allows all three clock modes.)

Table 2-3. Relationships Between Input Frequency, Clock Multiplier, and State Times

F_{XTAL1} (Frequency on XTAL1)	PLLEN2:1	Multiplier	f (Input Frequency to the Divide-by-two Circuit)	t (Clock Period)	State Time
50 MHz †	00	1	50 MHz	20 ns	40 ns
25 MHz	00	1	25 MHz	40 ns	80 ns
	10	2	50 MHz	20 ns	40 ns
12.5 MHz	00	1	12.5 MHz	80 ns	160 ns
	10	2	25 MHz	40 ns	80 ns
	11	4	50 MHz	20 ns	40 ns

† Assumes an external clock. The maximum frequency for an external crystal oscillator is 25 MHz.

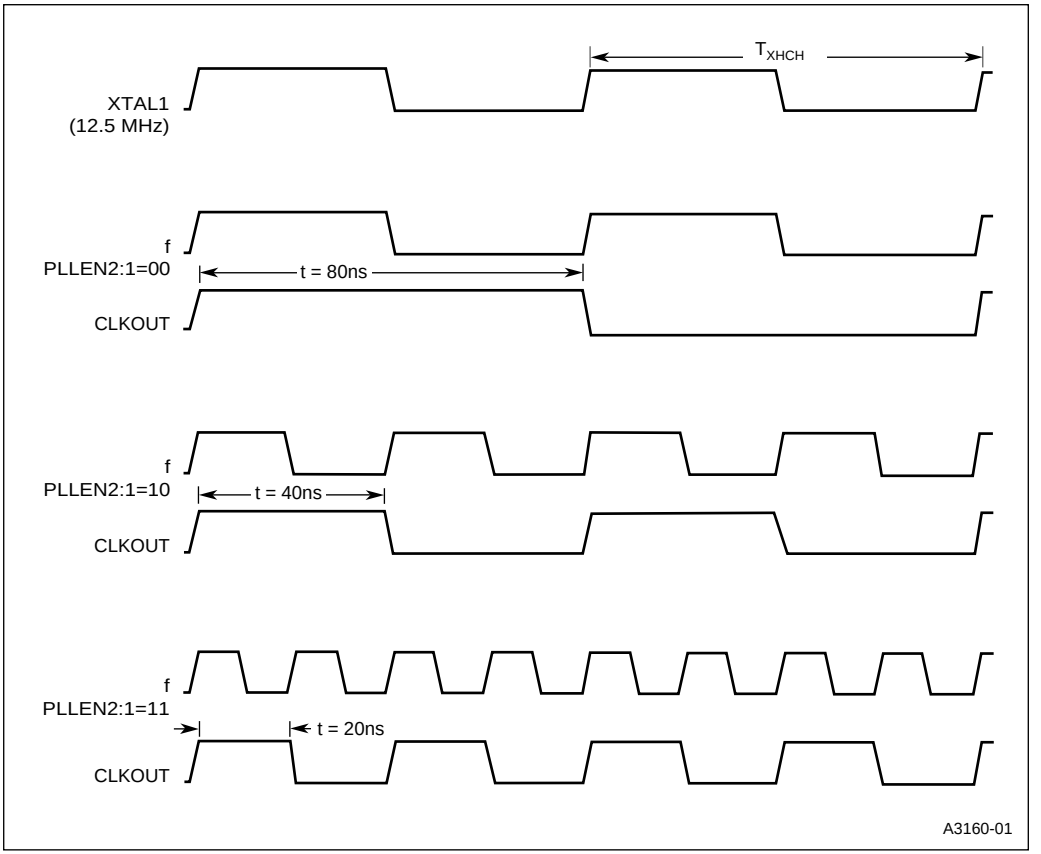


Figure 2-6. Effect of Clock Mode on CLKOUT Frequency

2.5 INTERNAL PERIPHERALS

The internal peripheral modules provide special functions for a variety of applications. This section provides a brief description of the peripherals; subsequent chapters describe them in detail.

2.5.1 I/O Ports

The 8XC196NP and 80C196NU have five I/O ports, ports 1–4 and the EPORT. Individual port pins are multiplexed to serve as standard I/O or to carry special-function signals associated with an on-chip peripheral or an off-chip component. If a particular special-function signal is not used in an application, the associated pin can be individually configured to serve as a standard I/O pin. Port 4 has a higher drive capability than the other ports to support pulse-width modulator (PWM) high-drive outputs.

Ports 1–4 are eight-bit, bidirectional, standard I/O ports. Only the lower nibble of port 4 is implemented in current package offerings. Port 1 provides I/O pins for the four event processor array (EPA) modules and the two timers. Port 2 is used for the serial I/O (SIO) port, two external interrupts, and bus hold functions. Port 3 is used for chip-select functions and two external interrupts. Port 4 (functionally only a 4-bit port) provides I/O pins associated with the three on-chip pulse-width modulators. The EPORT provides address lines A19:16 to support extended addressing. See Chapter 7, “I/O Ports,” for more information.

2.5.2 Serial I/O (SIO) Port

The serial I/O (SIO) port is an asynchronous/synchronous port that includes a universal asynchronous receiver and transmitter (UART). The UART has one synchronous mode (mode 0) and three asynchronous modes (modes 1, 2, and 3) for both transmission and reception. The asynchronous modes are full duplex, meaning that they can transmit and receive data simultaneously. The receiver is buffered, so the reception of a second byte can begin before the first byte is read. The transmitter is also buffered, allowing continuous transmissions. See Chapter 8, “Serial I/O (SIO) Port,” for details.

2.5.3 Event Processor Array (EPA) and Timer/Counters

The event processor array (EPA) performs high-speed input and output functions associated with its timer/counters. In the input mode, the EPA monitors an input for signal transitions. When an event occurs, the EPA records the timer value associated with it. This is a *capture* event. In the output mode, the EPA monitors a timer until its value matches that of a stored time value. When a match occurs, the EPA triggers an output event, which can set, clear, or toggle an output pin. This is a *compare* event. Both capture and compare events can initiate interrupts, which can be serviced by either the interrupt controller or the PTS.

Timer 1 and timer 2 are both 16-bit up/down timer/counters that can be clocked internally or externally. Each timer/counter is called a *timer* if it is clocked internally and a *counter* if it is clocked externally. See Chapter 10, "Event Processor Array (EPA)," for additional information on the EPA and timer/counters.

2.5.4 Pulse-width Modulator (PWM)

The output waveform from each PWM channel is a variable duty-cycle pulse with a programmable frequency that occurs every 256 or 512 state times (for the 8XC196NP) or every 256, 512, or 1024 state times (for the 80C196NU), as programmed. Several types of motors require a PWM waveform for most efficient operation. When filtered, the PWM waveform produces a DC level that can change in 256 steps by varying the duty cycle. See Chapter 9, "Pulse-width Modulator," for more information.

2.6 SPECIAL OPERATING MODES

In addition to the normal execution mode, the device operates in several special-purpose modes. Idle and powerdown modes conserve power when the device is inactive. An additional power conservation mode, standby, is available on the 80C196NU. On-circuit emulation (ONCE) mode electrically isolates the microcontroller from the system. See Chapter 12, "Special Operating Modes," for more information about idle, powerdown, standby, and ONCE modes.

2.6.1 Reducing Power Consumption

The power saving modes selectively disable internal clocks to reduce power consumption. Figure 2-3 on page 2-7 and Figure 2-4 on page 2-8 illustrate the clock circuitry of the 8XC196NP and 80C196NU, respectively.

In idle mode, the CPU stops executing instructions, but the peripheral clocks remain active. Power consumption drops to about 40% of normal execution mode consumption. Either a hardware reset or any enabled interrupt source will bring the device out of idle mode.

The 80C196NU has an additional power saving mode, standby. In standby mode, all internal clocks are frozen at logic state zero, but the oscillator and phase-locked loop continue to run. Power consumption drops to about 10% of normal execution mode consumption. Either a hardware reset or any enabled external interrupt source will bring the device out of standby mode.

In powerdown mode, all internal clocks are frozen at logic state zero and the oscillator is shut off. The register file and most peripherals retain their data if V_{CC} is maintained. Power consumption drops into the µW range.

2.6.2 Testing the Printed Circuit Board

The on-circuit emulation (ONCE) mode electrically isolates the 8XC196 device from the system. By invoking ONCE mode, you can test the printed circuit board while the device is soldered onto the board.

2.7 DESIGN CONSIDERATIONS FOR 80C196NP TO 80C196NU CONVERSIONS

This section summarizes differences to consider when converting your design requirements from the 80C196NP to the 80C196NU.

- The 80C196NU can achieve an operating frequency of 50 MHz, while the 80C196NP can achieve only 25 MHz.
- The 80C196NU is pin-compatible with the 80C196NP. The functions of four pins differ:
  - — the 80C196NU has PLLEN1 in place of a no-connection pin of the 80C196NP
  - — the 80C196NU has PLLEN2 in place of a V_{SS} pin of the 80C196NP
  - — the 80C196NU has a V_{CC} pin in place of a no-connection pin of the 80C196NP
  - — the 80C196NU has a no-connection pin in place of the EA# pin of the 80C196NP
- The 80C196NU requires that you tie the PLLEN1 and PLLEN2 pins either high or low, depending on the clock multiplier mode you select.
- The 80C196NU requires that you connect an external capacitor to the RPD pin if your design uses both powerdown mode and a clock multiplier mode.
- The 80C196NU has a new, 32-bit accumulator register and an accumulator status register to support its multiply-accumulate functions.
- The 80C196NU, since it has no nonvolatile memory, has no REMAP bit in the CCB.
- The 80C196NU can window additional memory into the lower register file via a second window selection register (WSR1).
- Unlike the 80C196NP, the 80C196NU's EPORT special-function registers are located in SFR address space, rather than in memory-mapped space, so they can be windowed for direct access.
- The 80C196NU has an 8-byte prefetch queue, while the 80C196NP has a 4-byte prefetch queue.
- In the 80C196NU, data accesses have a higher priority than instruction queue fetches. In the 80C196NP, the opposite is true (instruction fetches have the highest priority).
- The 80C196NU's serial I/O port has a divide-by-2 prescaler, controlled by the SP_CON register.
- The 80C196NU's EPA has an additional prescaler option (divide-by-128), controlled by the timer control register (T*x*_CONTROL).

- The 80C196NU's PWM has an additional prescaler option (divide-by-4), controlled by the PWM control register (CON_REG0).
- When operating with a demultiplexed bus, the 80C196NU can add an automatic delay in the first cycle following a chip-select change or in a write cycle that follows a read. This mode, called *deferred mode*, extends the following timing specifications by two clock periods (2t): T_{AVDV}, T_{AVWL}, T_{AVRL}, T_{RLDV}, T_{RHDZ}, T_{RHRL}, T_{LHLH}, T_{RHLH}, T_{SLDV}, and T_{WHLH}.
- The 80C196NU has an additional power-saving mode, standby (IDLPD #3).
- The 8XC196NP allows you to change the value of EP_REG to control which memory page a nonextended instruction accesses. However, software tools require that EP_REG be equal to 00H. The 80C196NU forces all nonextended data accesses to page 00H. You cannot use EP_REG to change pages.
- After a HOLD request, the 80C196NU's chip-select channels become inactive before the 80C196NU asserts HLDA#.
- In demultiplexed mode, the 80C196NU's RD# and WR# signals are asserted one clock period (1t) earlier than on the 80C196NP.

3

Advanced Math Features

CHAPTER 3
ADVANCED MATH FEATURES

The 80C196NU is the first member of the MCS® 96 microcontroller family to incorporate enhanced 16-bit multiplication instructions for performing multiply-accumulate operations and a dedicated, 32-bit accumulator register for storing the results of these operations. The accumulator and the enhanced instructions combine to decrease the amount of time required to perform multiply-accumulate operations. The instructions and accumulator support signed and unsigned integers as well as signed fractional data. This chapter describes the 80C196NU's advanced mathematical features.

3.1 ENHANCED MULTIPLICATION INSTRUCTIONS

The 16-bit multiplication instructions, MULU and MUL, that exist for all MCS 96 microcontrollers have been enhanced for the 80C196NU. The MULU instruction supports unsigned integers, while the MUL instruction supports signed integers and signed fractionals.

When you execute a 16-bit multiplication instruction with a destination address that is 0FH or below, the 80C196NU automatically stores the result in the accumulator. If bit 3 of the destination address is set (address 08H, 09H, …, 0FH), the 80C196NU clears the accumulator before it stores the result of the current instruction. If bit 3 of the destination address is clear (address 00H, 01H, …, 07H), it adds the result of the current instruction to the existing contents of the accumulator.

This simple example illustrates the results of consecutive multiply-accumulate instructions. The results of the first three instructions are automatically added together in the accumulator, while the last instruction clears the accumulator before the result is stored.

```
register_1 = 10 decimal (0AH),register_2 = 20 decimal (14H)
register_3 = 30 decimal (1EH),register_4 = 40 decimal (28H)

mul 00H,register_1,register_2 ;10×20= 200. Accumulator = 200 decimal.
mul 00H,register_3,register_4 ;30×40=1200. Accumulator =1400 decimal.
mul 00H,register_2,register_4 ;20×40= 800. Accumulator =2200 decimal.
mul 08H,register_2,register_3 ;20×30= 600. Accumulator = 600 decimal.
```

Table 3-1 compares the instructions required to perform a multiply-accumulate operation for the 8XC196NP and those required for the 80C196NU. The 8XC196NP requires four instructions, while the 80C196NU requires only one to accomplish the same operation. The four 8XC196NP instructions take a total of 32 state times to execute, while the single 80C196NU instruction takes only 16 state times. In addition, the 80C196NU can operate at twice the frequency of the 8XC196NP; therefore, a state time for the 80C196NU is half that of the 8XC196NP. These two factors combine to make the 80C196NU code execute in one-fourth the time required for the 8XC196NP code.

Table 3-1. Multiply/Accumulate Example Code

Device	Instructions		Execution Time	
8XC196NP (25 MHz; 1 state time = 80 ns)	mul	temp,operand_2,operand_1	16 states	1280 ns
	shll	temp,#1	8 states	640 ns
	add	out_l,temp_l	4 states	320 ns
	addc	out_h,temp_h	4 states	320 ns
			32 states total	2560 ns total
80C196NU (50 MHz; 1 state time = 40 ns)	mul	08H,operand_2,operand_1†	16 states	640 ns
			16 states total	640 ns total

† Because bit 3 of the destination address (08H) is set, the 80C196NU clears the accumulator before adding the result of the current instruction to it. If bit 3 were clear (destination address 07H–00H), the 80C196NU would add the result of the current instruction to the existing value of the accumulator.

3.2 OPERATING MODES

The accumulator has two operating modes that allow you to control the results of operations on signed numbers. These modes are called *saturation mode* and *fractional mode*.

3.2.1 Saturation Mode

Saturation occurs when the result of two positive numbers generates a negative sign bit or the result of two negative numbers generates a positive sign bit. Without saturation mode, an underflow or overflow occurs and the overflow (OVF) flag is set. Saturation mode prevents an underflow or overflow of the accumulated value. In saturation mode, the accumulator's value is changed to 7FFFFFFFH for a positive saturation or 80000000H for a negative saturation and the sticky saturation (STSAT) flag is set. The following two examples illustrate the contents of the accumulator as a result of positive and negative saturation, respectively:

7FFFFFFFH = 0111 1111 1111 1111 1111 1111 1111 1111 = $2^{31} - 1$ = +2147483647

80000000H = 1000 0000 0000 0000 0000 0000 0000 0000 = –2147483648

3.2.2 Fractional Mode

A *signed fractional* contains an imaginary decimal point between the sign bit (the MSB) and the adjacent bit. These examples illustrate the representation of 32-bit signed fractional numbers:

$$0.111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111 = \frac{2147483647}{2147483648} = 1$$

$$0.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 = 0$$

$$1.111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111 = \frac{-1}{2147483648} = -0$$

$$1.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 = -1$$

Fractional mode shifts the result of a multiplication instruction left by one bit before writing the result to the accumulator. This left shift eliminates the extra sign bit when both operands are signed, leaving a correctly signed result and the correct decimal placement.

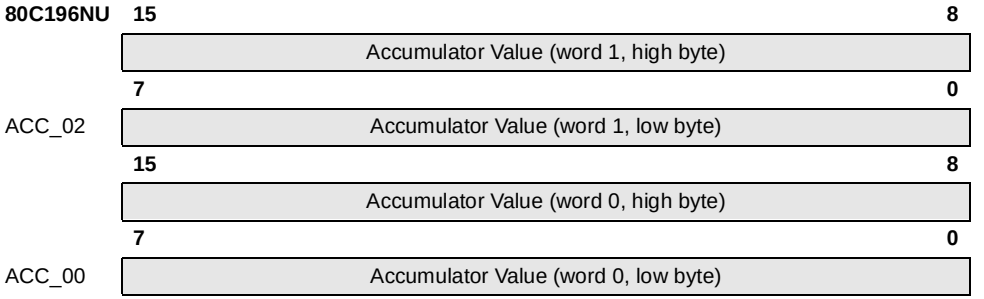
3.3 ACCUMULATOR REGISTER (ACC_0*x*)

The 32-bit accumulator register (Figure 3-1) resides at locations 0C–0FH. Read from or write to the accumulator register as two words at locations 0CH and 0EH.

ACC_0*x*
***x* = 0, 2 (80C196NU)**

Address: 0EH, 0CH
Reset State: 00H

The 32-bit accumulator register (ACC_0*x*) resides at locations 0C–0FH. You can read from or write to the accumulator register as two words at locations 0CH and 0EH.



Bit Number	Function
15:0	Accumulator Value You can read this register to determine the current value of the accumulator. You can write to this register to clear or preload a value into the accumulator.

Figure 3-1. Accumulator (ACC_0*x*) Register

3.4 ACCUMULATOR CONTROL AND STATUS REGISTER (ACC_STAT)

The ACC_STAT register controls the operating mode and reflects the status of the accumulator. The mode bits (FME and SME) are effective only for signed multiplication. Table 3-2 describes the 80C196NU's operation with each of the four possible configurations of these bits.

ACC_STAT (80C196NU) Address: 0BH Reset State: 00H

The accumulator control and status (ACC_STAT) register enables and disables fractional and saturation modes and contains three status flags that indicate the status of the accumulator's contents.

	7							0
80C196NU	FME	SME	—	—	—	STOVF	OVF	STSAT

Bit Number	Bit Mnemonic	Function
7	FME	Fractional Mode Enable Set this bit to enable fractional mode. (See Table 3-2.) In this mode, the result of a signed multiplication instruction is shifted left by one bit before it is added to the contents of the accumulator. For unsigned multiplication, this bit is ignored.
6	SME	Saturation Mode Enable Set this bit to enable saturation mode. (See Table 3-2.) In this mode, the result of a signed multiplication operation is **not** allowed to overflow or underflow. For unsigned multiplication, this bit is ignored.
5:3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	STOVF	Sticky Overflow Flag For unsigned multiplication, this bit is set if a carry out of bit 31 occurs. Unless saturation mode is enabled, this bit is set for signed multiplication to indicate that the sign bit of the accumulator and the sign bit of the addend are equal, but the sign bit of the result is the opposite. (See Table 3-2.) Software can clear this flag; hardware does **not** clear it.
1	OVF	Overflow Flag This bit indicates that an overflow occurred during the preceding accumulation. (See Table 3-2.) This flag is dynamic; it can change after each accumulation.
0	STSAT	Sticky Saturation Flag This bit indicates that a saturation has occurred during accumulation with saturation mode enabled. (See Table 3-2.) Software can clear this flag; hardware does **not** clear it.

Figure 3-2. Accumulator Control and Status (ACC_STAT) Register

Table 3-2. Effect of SME and FME Bit Combinations

SME	FME	Description
0	0	Sets the OVF and STOVF flags if the sign bits of the accumulator and the addend (the number to be added to the contents of the accumulator) are equal, but the sign bit of the result is the opposite.
0	1	Shifts the addend (the number to be added to the contents of the accumulator) left by one bit before adding it to the accumulator. Sets the OVF and STOVF flags if the sign bits of the accumulator and the addend are equal, but the sign bit of the result is the opposite.
1	0	Accumulates a signed integer value up or down to saturation and sets the STSAT flag. Positive saturation changes the accumulator value to 7FFFFFFFH; negative saturation changes the accumulator value to 80000000H. Accumulation proceeds normally after saturation, which means that the accumulator value can increase from a negative saturation or decrease from a positive saturation.
1	1	Shifts the addend (the number to be added to the contents of the accumulator) left by one bit before adding it to the accumulator. Accumulates a signed integer value up or down to saturation and sets the STSAT flag. Positive saturation changes the accumulator value to 7FFFFFFFH; negative saturation changes the accumulator value to 80000000H. Accumulation proceeds normally after saturation, which means that the accumulator value can increase from a negative saturation or decrease from a positive saturation.

4

Programming Considerations

CHAPTER 4
PROGRAMMING CONSIDERATIONS

This section provides an overview of the instruction set of the MCS® 96 microcontrollers and offers guidelines for program development. For detailed information about specific instructions, see Appendix A.

4.1 OVERVIEW OF THE INSTRUCTION SET

The instruction set supports a variety of operand types likely to be useful in control applications (see Table 4-1).

NOTE

The operand-type variables are shown in all capitals to avoid confusion. For example, a *BYTE* is an unsigned 8-bit variable in an instruction, while a *byte* is any 8-bit unit of data (either signed or unsigned).

Table 4-1. Operand Type Definitions

Operand Type	No. of Bits	Signed	Possible Values	Addressing Restrictions
BIT	1	No	True (1) or False (0)	As components of bytes
BYTE	8	No	0 through 2^8–1 (0 through 255)	None
SHORT-INTEGER	8	Yes	-2^7 through $+2^7$–1 (–128 through +127)	None
WORD	16	No	0 through 2^{16}–1 (0 through 65,535)	Even byte address
INTEGER	16	Yes	-2^{15} through $+2^{15}$–1 (–32,768 through +32,767)	Even byte address
DOUBLE-WORD (Note 1)	32	No	0 through 2^{32}–1 (0 through 4,294,967,295)	An address in the lower register file that is evenly divisible by four (Note 2)
LONG-INTEGER (Note 1)	32	Yes	-2^{31} through $+2^{31}$–1 (–2,147,483,648 through +2,147,483,647)	An address in the lower register file that is evenly divisible by four (Note 2)
QUAD-WORD (Note 3)	64	No	0 through 2^{64}–1	An address in the lower register file that is evenly divisible by eight

NOTES:
1. The 32-bit variables are supported only as the operand in shift operations, as the dividend in 32-by-16 divide operations, and as the product of 16-by-16 multiply operations.
2. For consistency with third-party software, you should adopt the C programming conventions for addressing 32-bit operands. For more information, refer to page 4-11.
3. QUAD-WORD variables are supported only as the operand for the EBMOVI instruction.

Table 4-2 lists the equivalent operand-type names for both C programming and assembly language.

Table 4-2. Equivalent Operand Types for Assembly and C Programming Languages

Operand Types	Assembly Language Equivalent	C Programming Language Equivalent
BYTE	BYTE	unsigned char
SHORT-INTEGER	BYTE	char
WORD	WORD	unsigned int
INTEGER	WORD	int
DOUBLE-WORD	LONG	unsigned long
LONG-INTEGER	LONG	long
QUAD-WORD	—	—

4.1.1 BIT Operands

A BIT is a single-bit variable that can have the Boolean values, "true" and "false." The architecture requires that BITs be addressed as components of BYTEs or WORDs. It does not support the direct addressing of BITs.

4.1.2 BYTE Operands

A BYTE is an unsigned, 8-bit variable that can take on values from 0 through 255 (2^8–1). Arithmetic and relational operators can be applied to BYTE operands, but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTEs are applied bitwise. Bits within BYTEs are labeled from 0 to 7; bit 0 is the least-significant bit. There are no alignment restrictions for BYTEs, so they may be placed anywhere in the address space.

4.1.3 SHORT-INTEGER Operands

A SHORT-INTEGER is an 8-bit, signed variable that can take on values from –128 ($–2^7$) through +127 ($+2^7$–1). Arithmetic operations that generate results outside the range of a SHORT-INTEGER set the overflow flags in the processor status word (PSW). The numeric result is the same as the result of the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERs, so they may be placed anywhere in the address space.

4.1.4 WORD Operands

A WORD is an unsigned, 16-bit variable that can take on values from 0 through 65,535 (2^{16}–1). Arithmetic and relational operators can be applied to WORD operands, but the result must be interpreted in modulo 65536 arithmetic. Logical operations on WORDs are applied bitwise. Bits within WORDs are labeled from 0 to 15; bit 0 is the least-significant bit.

WORDs must be aligned at even byte boundaries in the address space. The least-significant byte of the WORD is in the even byte address, and the most-significant byte is in the next higher (odd) address. The address of a WORD is that of its least-significant byte (the even byte address). WORD operations to odd addresses are not guaranteed to operate in a consistent manner.

4.1.5 INTEGER Operands

An INTEGER is a 16-bit, signed variable that can take on values from –32,768 (-2^{15}) through +32,767 ($+2^{15}$–1). Arithmetic operations that generate results outside the range of an INTEGER set the overflow flags in the processor status word (PSW). The numeric result is the same as the result of the equivalent operation on WORD variables.

INTEGERs must be aligned at even byte boundaries in the address space. The least-significant byte of the INTEGER is in the even byte address, and the most-significant byte is in the next higher (odd) address. The address of an INTEGER is that of its least-significant byte (the even byte address). INTEGER operations to odd addresses are not guaranteed to operate in a consistent manner.

4.1.6 DOUBLE-WORD Operands

A DOUBLE-WORD is an unsigned, 32-bit variable that can take on values from 0 through 4,294,967,295 (2^{32}–1). The architecture directly supports DOUBLE-WORD operands only as the operand in shift operations, as the dividend in 32-by-16 divide operations, and as the product of 16-by-16 multiply operations. For these operations, a DOUBLE-WORD variable must reside in the lower register file and must be aligned at an address that is evenly divisible by four. The address of a DOUBLE-WORD is that of its least-significant byte (the even byte address). The least-significant word of the DOUBLE-WORD is always in the lower address, even when the data is in the stack. This means that the most-significant word must be pushed into the stack first.

DOUBLE-WORD operations that are not directly supported can be easily implemented with two WORD operations. For example, the following sequences of 16-bit operations perform a 32-bit addition and a 32-bit subtraction, respectively.

```
ADD  REG1,REG3                    ; (2-operand addition)
ADDC REG2,REG4

SUB  REG1,REG3                    ; (2-operand subtraction)
SUBC REG2,REG4
```

4.1.7 LONG-INTEGER Operands

A LONG-INTEGER is a 32-bit, signed variable that can take on values from –2,147,483,648 (-2^{31}) through +2,147,483,647 ($+2^{31}-1$). The architecture directly supports LONG-INTEGER operands only as the operand in shift operations, as the dividend in 32-by-16 divide operations, and as the product of 16-by-16 multiply operations. For these operations, a LONG-INTEGER variable must reside in the lower register file and must be aligned at an address that is evenly divisible by four. The address of a LONG-INTEGER is that of its least-significant byte (the even byte address).

LONG-INTEGER operations that are not directly supported can be easily implemented with two INTEGER operations. See the example in "DOUBLE-WORD Operands" on page 4-3.

4.1.8 QUAD-WORD Operands

A QUAD-WORD is a 64-bit, unsigned variable that can take on values from 0 through $2^{64}-1$. The architecture directly supports the QUAD-WORD operand only as the operand of the EBMOVI instruction. For this operation, the QUAD-WORD variable must reside in the lower register file and must be aligned at an address that is evenly divisible by eight.

4.1.9 Converting Operands

The instruction set supports conversions between the operand types. The LDBZE (load byte, zero extended) instruction converts a BYTE to a WORD. CLR (clear) converts a WORD to a DOUBLE-WORD by clearing (writing zeros to) the upper WORD of the DOUBLE-WORD. LDBSE (load byte, sign extended) converts a SHORT-INTEGER into an INTEGER. EXT (sign extend) converts an INTEGER to a LONG-INTEGER.

4.1.10 Conditional Jumps

The instructions for addition, subtraction, and comparison do not distinguish between unsigned (BYTE, WORD) and signed (SHORT-INTEGER, INTEGER) operands. However, the conditional jump instructions allow you to treat the results of these operations as signed or unsigned quantities. For example, the CMP (compare) instruction is used to compare both signed and unsigned 16-bit quantities. Following a compare operation, you can use the JH (jump if higher) instruction for unsigned operands or the JGT (jump if greater than) instruction for signed operands.

4.1.11 Floating Point Operations

The hardware does not directly support operations on REAL (floating point) variables. Those operations are supported by floating point libraries from third-party tool vendors. (See the *Development Tools Handbook*.) The performance of these operations is significantly improved by the NORML instruction and by the sticky bit (ST) flag in the processor status word (PSW). The NORML instruction normalizes a 32-bit variable; the sticky bit (ST) flag can be used in conjunction with the carry (C) flag to achieve finer resolution in rounding.

4.1.12 Extended Instructions

This section briefly describes the instructions that have been added to enable code execution and data access anywhere in the 1-Mbyte address space.

NOTE

In 1-Mbyte mode, ECALL, LCALL, and SCALL always push two words onto the stack; therefore, a RET must always pop two words from the stack. Because of the extra push and pop operations, interrupt routines and subroutines take slightly longer to execute in 1-Mbyte mode than in 64-Kbyte mode.

EBMOVI **Extended interruptable block move**. Moves a block of word data from one memory location to another. This instruction allows you to move blocks of up to 64K words between any two locations in the address space. It uses two 24-bit autoincrementing pointers and a 16-bit counter.

EBR **Extended branch**. This instruction is an unconditional indirect jump to anywhere in the address space. It functions only in extended addressing modes.

ECALL **Extended call**. This instruction is an unconditional relative call to anywhere in the address space. It functions only in extended addressing modes.

EJMP **Extended jump**. This instruction is an unconditional, relative jump to anywhere in the address space. It functions only in extended addressing modes.

ELD **Extended load word**. Loads the value of the source word operand into the destination operand. This instruction allows you to move data from anywhere in the address space into the lower register file. It operates in extended indirect and extended indexed modes.

ELDB **Extended load byte**. Loads the value of the source byte operand into the destination operand. This instruction allows you to move data from anywhere in the address space into the lower register file. It operates in extended indirect and extended indexed modes.

EST **Extended store word**. Stores the value of the source **(leftmost)** word operand into the destination **(rightmost)** operand. This instruction allows you to move data from the lower register file to anywhere in the address space. It operates in extended indirect and extended indexed modes.

ESTB **Extended store byte**. Stores the value of the source **(leftmost)** byte operand into the destination **(rightmost)** operand. This instruction allows you to move data from the lower register file to anywhere in the address space. It operates in extended indirect and extended indexed modes.

4.2 ADDRESSING MODES

The instruction set uses four basic addressing modes:

- direct
- immediate
- indirect (with or without autoincrement)
- indexed (short-, long-, or zero-indexed)

The stack pointer can be used with indirect addressing to access the top of the stack, and it can also be used with short-indexed addressing to access data within the stack. The zero register can be used with long-indexed addressing to access any memory location.

Extended variations of the indirect and indexed modes support the extended load and store instructions. An extended load instruction moves a word (ELD) or a byte (ELDB) from any location in the address space into the lower register file. An extended store instruction moves a word (EST) or a byte (ESTB) from the lower register file into any location in the address space. An instruction can contain only one immediate, indirect, or indexed reference; any remaining operands must be direct references.

This section describes the addressing modes as they are handled by the hardware. An understanding of these details will help programmers to take full advantage of the architecture. The assembly language hides some of the details of how these addressing modes work. "Assembly Language Addressing Mode Selections" on page 4-11 describes how the assembly language handles direct and indexed addressing modes.

The examples in this section assume that temporary registers are defined as shown in this segment of assembly code and described in Table 4-3.

```
          Oseg at 1ch
AX        DSW  1
BX        DSW  1
CX        DSW  1
DX        DSW  1
EX        DSL  1
```

Table 4-3. Definition of Temporary Registers

Temporary Register	Description
AX	word-aligned 16-bit register; AH is the high byte of AX and AL is the low byte
BX	word-aligned 16-bit register; BH is the high byte of BX and BL is the low byte
CX	word-aligned 16-bit register; CH is the high byte of CX and CL is the low byte
DX	word-aligned 16-bit register; DH is the high byte of DX and DL is the low byte
EX	double-word-aligned 24-bit register

4.2.1 Direct Addressing

Direct addressing directly accesses a location in the 256-byte lower register file, without involving the memory controller. Windowing allows you to remap other sections of memory into the lower register file for direct access (see Chapter 5, “Memory Partitions,” for details). You specify the registers as operands within the instruction. The register addresses must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in a calculation. The following instructions use direct addressing:

```
ADD  AX,BX,CX          ; AX ← BX + CX
ADDB AL,BL,CL          ; AL ← BL + CL
MUL  AX,BX             ; AX ← AX × BX
INCB CL                ; CL ← CL + 1
```

4.2.2 Immediate Addressing

Immediate addressing mode accepts one immediate value as an operand in the instruction. You specify an immediate value by preceding it with a number symbol (#). An instruction can contain only one immediate value; the remaining operands must be direct references. The following instructions use immediate addressing:

```
ADD  AX,#340       ; AX ← AX + 340
PUSH #1234H        ; SP ← SP - 2
                   ; MEM_WORD(SP) ← 1234H
DIVB AX,#10        ; AL ← AX/10
                   ; AH ← AX MOD 10
```

4.2.3 Indirect Addressing

The indirect addressing mode accesses an operand by obtaining its address from a WORD register in the lower register file. You specify the register containing the indirect address by enclosing it in square brackets ([ ]). The indirect address can refer to any location within the address space, including the register file. The register that contains the indirect address must be word-aligned, and the indirect address must conform to the rules for the operand type. An instruction can contain only one indirect reference; any remaining operands must be direct references. The following instructions use indirect addressing:

```
LD   AX,[BX]              ; AX ← MEM_WORD(BX)
ADDB AL,BL,[CX]           ; AL ← BL + MEM_BYTE(CX)
POP  [AX]                 ; MEM_WORD(AX) ← MEM_WORD(SP)
                          ; SP ← SP + 2
```

4.2.3.1 Extended Indirect Addressing

Extended load and store instructions can use indirect addressing. The only difference is that the register containing the indirect address must be a word-aligned 24-bit register to allow access to the entire 1-Mbyte address space. The following instructions use extended indirect addressing:

```
ELD  AX,[EX]         ; AX ← MEM_WORD(EX)
ELDB AL,[EX]         ; AL ← MEM_BYTE(EX)
EST  AX,[EX]         ; MEM_WORD(EX) ← AX
ESTB AL,[EX]         ; MEM_BYTE(EX) ← AL
```

4.2.3.2 Indirect Addressing with Autoincrement

You can choose to automatically increment the indirect address after the current access. You specify autoincrementing by adding a plus sign (+) to the end of the indirect reference. In this case, the instruction automatically increments the indirect address (by one if the destination is an 8-bit register or by two if it is a 16-bit register). When your code is assembled, the assembler automatically sets the least-significant bit of the indirect address register. The following instructions use indirect addressing with autoincrement:

```
LD   AX,[BX]+        ; AX ← MEM_WORD(BX)
                     ; BX ← BX + 2
ADDB AL,BL,[CX]+     ; AL ← BL + MEM_BYTE(CX)
                     ; CX ← CX + 1
PUSH [AX]+           ; SP ← SP - 2
                     ; MEM_WORD(SP) ← MEM_WORD(AX)
                     ; AX ← AX + 2
```

4.2.3.3 Extended Indirect Addressing with Autoincrement

The extended load and store instructions can also use indirect addressing with autoincrement. The only difference is that the register containing the indirect address must be a word-aligned 24-bit register to allow access to the entire 1-Mbyte address space. The following instructions use extended indirect addressing with autoincrement:

```
ELD  AX,[EX]+        ; AX ← MEM_WORD(EX)
                     ; EX ← EX + 2
ELDB AL,[EX]+        ; AL ← MEM_BYTE(EX)
                     ; EX ← EX + 2
EST  AX,[EX]+        ; MEM_WORD(EX) ← AX
                     ; MEM_WORD(EX) ← MEM_WORD(EX + 2)
ESTB AL,[EX]+        ; MEM_BYTE(EX) ← AL
                     ; MEM_BYTE(EX) ← MEM_BYTE(EX + 2)
```

4.2.3.4 Indirect Addressing with the Stack Pointer

You can also use indirect addressing to access the top of the stack by using the stack pointer as the WORD register in an indirect reference. The following instruction uses indirect addressing with the stack pointer:

```
PUSH [SP]       ; duplicate top of stack
                ; SP ← SP +2
```

4.2.4 Indexed Addressing

Indexed addressing calculates an address by adding an offset to a base address. There are three variations of indexed addressing: short-indexed, long-indexed, and zero-indexed. Both short- and long-indexed addressing are used to access a specific element within a structure. Short-indexed addressing can access up to 255 byte locations, long-indexed addressing can access up to 65,535 byte locations, and zero-indexed addressing can access a single location. An instruction can contain only one indexed reference; any remaining operands must be direct references.

4.2.4.1 Short-indexed Addressing

In a short-indexed instruction, you specify the offset as an 8-bit constant and the base address as an indirect address register (a WORD). The following instructions use short-indexed addressing.

```
LD   AX,12H[BX]      ; AX ← MEM_WORD(BX+12H)
MULB AX,BL,3[CX]     ; AX ← BL ×MEM_BYTE(CX+3)
```

The instruction LD AX,12H[BX] loads AX with the contents of the memory location that resides at address BX+12H. That is, the instruction adds the constant 12 (the offset) to the contents of BX (the base address), then loads AX with the contents of the resulting address. For example, if BX contains 1000H, then AX is loaded with the contents of location 1012H. Short-indexed addressing is typically used to access elements in a structure, where BX contains the base address of the structure and the constant (12H in this example) is the offset of a specific element in a structure.

You can also use the stack pointer in a short-indexed instruction to access a particular location within the stack, as shown in the following instruction.

```
LD   AX,2[SP]
```

4.2.4.2 Long-indexed Addressing

In a long-indexed instruction, you specify the base address as a 16-bit variable and the offset as an indirect address register (a WORD). The following instructions use long-indexed addressing.

```
LD   AX,TABLE[BX]        ; AX ← MEM_WORD(TABLE+BX)
AND  AX,BX,TABLE[CX]     ; AX ← BX AND MEM_WORD(TABLE+CX)
```

```
ST   AX,TABLE[BX]         ; MEM_WORD(TABLE+BX) ← AX
ADDB AL,BL,LOOKUP[CX]     ; AL ← BL + MEM_BYTE(LOOKUP+CX)
```

The instruction LD AX, TABLE[BX] loads AX with the contents of the memory location that resides at address TABLE+BX. That is, the instruction adds the contents of BX (the offset) to the constant TABLE (the base address), then loads AX with the contents of the resulting address. For example, if TABLE equals 4000H and BX contains 12H, then AX is loaded with the contents of location 4012H. Long-indexed addressing is typically used to access elements in a table, where TABLE is a constant that is the base address of the structure and BX is the scaled offset ($n \times$ element size, in bytes) into the structure.

4.2.4.3 Extended Indexed Addressing

The extended load and store instructions can use extended indexed addressing. The only difference from long-indexed addressing is that both the base address and the offset must be 24 bits to support access to the entire 1-Mbyte address space. The following instructions use extended indexed addressing. (In these instructions, OFFSET is a 24-bit variable containing the offset, and EX is a double-word aligned 24-bit register containing the base address.)

```
ELD  AX,OFFSET[EX]        ; AX ← MEM_WORD(EX+OFFSET)
ELDB AL,OFFSET[EX]        ; AL ← MEM_BYTE(EX+OFFSET)
EST  AX,OFFSET[EX]        ; MEM_WORD(EX+OFFSET) ← AX
ESTB AL,OFFSET[EX]        ; MEM_BYTE(EX+OFFSET) ← AL
```

4.2.4.4 Zero-indexed Addressing

In a zero-indexed instruction, you specify the address as a 16-bit variable; the offset is zero, and you can express it in one of three ways: [0], [ZERO_REG], or nothing. Each of the following load instructions loads AX with the contents of the variable THISVAR.

```
LD   AX,THISVAR[0]
LD   AX,THISVAR[ZERO_REG]
LD   AX,THISVAR
```

The following instructions also use zero-indexed addressing:

```
ADD  AX,1234H[ZERO_REG]       ; AX ← AX + MEM_WORD(1234H)
POP  5678H[ZERO_REG]          ; MEM_WORD(5678H) ← MEM_WORD(SP)
                              ; SP ← SP + 2
```

4.2.4.5 Extended Zero-indexed Addressing

The extended instructions can also use zero-indexed addressing. The only difference is that you specify the address as a 24-bit constant or variable. The following extended instruction uses zero-indexed addressing. ZERO_REG acts as a 32-bit fixed source of the constant zero for an extended indexed reference.

```
ELD  AX,23456H[ZERO_REG]      ; AX ← MEM_WORD(23456H)
```

4.3 ASSEMBLY LANGUAGE ADDRESSING MODE SELECTIONS

The assembly language simplifies the choice of addressing modes. Use these features wherever possible.

4.3.1 Direct Addressing

The assembly language chooses between direct and zero-indexed addressing depending on the memory location of the operand. Simply refer to the operand by its symbolic name. If the operand is in the lower register file, the assembly language chooses a direct reference. If the operand is elsewhere in memory, it chooses a zero-indexed reference.

4.3.2 Indexed Addressing

The assembly language chooses between short-indexed and long-indexed addressing depending on the value of the index expression. If the value can be expressed in eight bits, the assembly language chooses a short-indexed reference. If the value is greater than eight bits, it chooses a long-indexed reference.

4.3.3 Extended Addressing

If the operand is outside page 00H, then you must use the extended load and store instructions, ELD, ELDB, EST, and ESTB.

4.4 DESIGN CONSIDERATIONS FOR 1-MBYTE DEVICES

In general, you should avoid creating tables or arrays that cross page boundaries. For example, if you are building a large array, start it at a base address that will accommodate the entire array within the same page. If you cannot avoid crossing a page boundary, keep in mind that you must use extended instructions to access data outside the original page.

4.5 SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size, it is a good idea to develop the program in modules and to establish standards that control communication between the modules. These standards vary with the needs of the final application. However, all standards must include some mechanism for passing parameters to procedures and returning results from procedures. We recommend that you use the conventions adopted by the C programming language for procedure linkage. These standards are usable for both the assembly language and C programming environments, and they offer compatibility between these environments.

4.5.1 Using Registers

The 256-byte lower register file contains the CPU special-function registers and the stack pointer. The remainder of the lower register file and all of the upper register file is available for your use. Peripheral special-function registers (SFRs) and memory-mapped SFRs reside in higher memory. The peripheral SFRs can be *windowed* into the lower register file for direct access. Memory-mapped SFRs cannot be windowed; you must use indirect or indexed addressing to access them. All SFRs can be operated on as BYTEs or WORDs, unless otherwise specified. See "Peripheral Special-function Registers (SFRs)" on page 5-7 and "Register File" on page 5-9 for more information.

To use these registers effectively, you must have some overall strategy for allocating them. The C programming language adopts a simple, effective strategy. It allocates the eight or sixteen bytes beginning at address 1CH as temporary storage and treats the remaining area in the register file as a segment of memory that is allocated as required.

NOTE

Using any SFR as a base or index register for indirect or indexed operations can cause unpredictable results because external events can change the contents of SFRs. Also, because some SFRs are cleared when read, consider the implications of using an SFR as an operand in a read-modify-write instruction (e.g., XORB).

4.5.2 Addressing 32-bit Operands

The 32-bit operands (DOUBLE-WORDs and LONG-INTEGERs) are formed by two adjacent 16-bit words in memory. The least-significant word of a DOUBLE-WORD is always in the lower address, even when the data is in the stack (which means that the most-significant word must be pushed into the stack first). The address of a 32-bit operand is that of its least-significant byte.

The hardware supports the 32-bit data types as operands in shift operations, as dividends of 32-by-16 divide operations, and as products of 16-by-16 multiply operations. For these operations, the 32-bit operand must reside in the lower register file and must be aligned at an address that is evenly divisible by four.

4.5.3 Addressing 64-bit Operands

The hardware supports the QUAD-WORD only as the operand of the EBMOVI instruction. For this operation, the QUAD-WORD variable must reside in the lower register file and must be aligned at an address that is evenly divisible by eight.

4.5.4 Linking Subroutines

Parameters are passed to subroutines via the stack. Parameters are pushed into the stack from the rightmost parameter to the left. The 8-bit parameters are pushed into the stack with the high-order byte undefined. The 32-bit parameters are pushed onto the stack as two 16-bit values; the most-significant half of the parameter is pushed into the stack first. As an example, consider the following procedure:

```
void example_procedure (char param1, long param2, int param3);
```

When this procedure is entered at run-time, the stack will contain the parameters in the following order:

```
param3
low word of param2
high word of param2
undefined;param1
return address       ← Stack Pointer
```

If a procedure returns a value to the calling code (as opposed to modifying more global variables) the result is returned in the temporary storage space (TMPREG0, in this example) starting at 1CH. TMPREG0 is viewed as either an 8-, 16-, 32-, or 64-bit variable, depending on the type of the procedure.

The standard calling convention adopted by the C programming language has several key features:

- Procedures can always assume that the eight or sixteen bytes of register file memory starting at 1CH can be used as temporary storage within the body of the procedure.
- Code that calls a procedure must assume that the procedure modifies the eight or sixteen bytes of register file memory starting at 1CH.
- Code that calls a procedure must assume that the procedure modifies the processor status word (PSW) condition flags because procedures do not save and restore the PSW.
- Function results from procedures are always returned in the variable TMPREG0.

The C programming language allows the definition of interrupt procedures, which are executed when a predefined interrupt request occurs. Interrupt procedures do not conform to the rules of normal procedures. Parameters cannot be passed to these procedures and they cannot return results. Since interrupt procedures can execute essentially at any time, they must save and restore both the PSW and TMPREG0.

4.6 SOFTWARE PROTECTION FEATURES AND GUIDELINES

The device has several features to assist in recovering from hardware and software errors. The unimplemented opcode interrupt provides protection from executing unimplemented opcodes. The hardware reset instruction (RST) can cause a reset if the program counter goes out of bounds. The RST instruction opcode is FFH, so the processor will reset itself if it tries to fetch an instruction from unprogrammed locations in nonvolatile memory or from bus lines that have been pulled high.

We recommend that you fill unused areas of code with NOPs and periodic jumps to an error routine or RST instruction. This is particularly important in the code surrounding lookup tables, since accidentally executing from lookup tables will cause undesired results. Wherever space allows, surround each table with seven NOPs (because the longest device instruction has seven bytes) and a RST or a jump to an error routine. Since RST is a one-byte instruction, the NOPs are unnecessary if RSTs are used instead of jumps to an error routine. This will help to ensure a speedy recovery from a software error.

5

Memory Partitions

CHAPTER 5
MEMORY PARTITIONS

This chapter describes the organization of the address space, its major partitions, and the 1-Mbyte and 64-Kbyte operating modes. *1-Mbyte* refers to the address space defined by the 20 external address lines. In 1-Mbyte mode, code can execute from almost anywhere in the 1-Mbyte space. In 64-Kbyte mode, code can execute only from the 64-Kbyte area FF0000–FFFFFFH. The 64-Kbyte mode provides compatibility with software written for previous 16-bit MCS® 96 microcontrollers. In either mode, nearly all of the 1-Mbyte address space is available for data storage.

Other topics covered in this chapter include the following:

- the relationship between the 1-Mbyte address space defined by the 20 external address lines and the 16-Mbyte address space defined by the 24 internal address lines
- extended and nonextended data accesses
- a *windowing* technique for accessing the upper register file and peripheral special-function registers (SFRs) with direct addressing
- examples of external memory configurations for the 1-Mbyte and 64-Kbyte operating modes
- a method for remapping the 4-Kbyte internal ROM (83C196NP only)

5.1 MEMORY MAP OVERVIEW

The instructions can address 16 Mbytes of memory. However, only 20 of the 24 address lines are implemented by external pins: A19:0 in demultiplexed mode, or A19:16 and AD15:0 in multiplexed mode. The lower 16 address/data lines, AD15:0, are the same as those in all other MCS 96 microcontrollers. The four extended address lines, A19:16, are provided by the EPORT. If, for example, an internal 24-bit address is FF2018H, the 20 external-address pins output F2018H. Further, the address seen by an external device depends on how many of the extended address lines are connected to the device. (See “Internal and External Addresses” on page 13-1.)

The 20 external-address pins can address 1 Mbyte of external memory. For purposes of discussion only, it is convenient to view this 1-Mbyte address space as sixteen 64-Kbyte pages, numbered 00H–0FH (see Figure 5-1 on page 5-2). The lower 16 address lines enable the device to address page 00H. The four extended address lines enable the device to address the remaining external address space, pages 01H–0FH.

Because the four most-significant bits (MSBs) of the internal address can take any values without changing the external address, these four bits effectively produce 16 copies of the 1-Mbyte address space, for a total of 16 Mbytes in 256 pages, 00H–FFH (Figure 5-1). For example, page 01H has 15 duplicates: 11H, 21H, ..., F1H. The shaded areas in Figure 5-1 represent the overlaid areas.

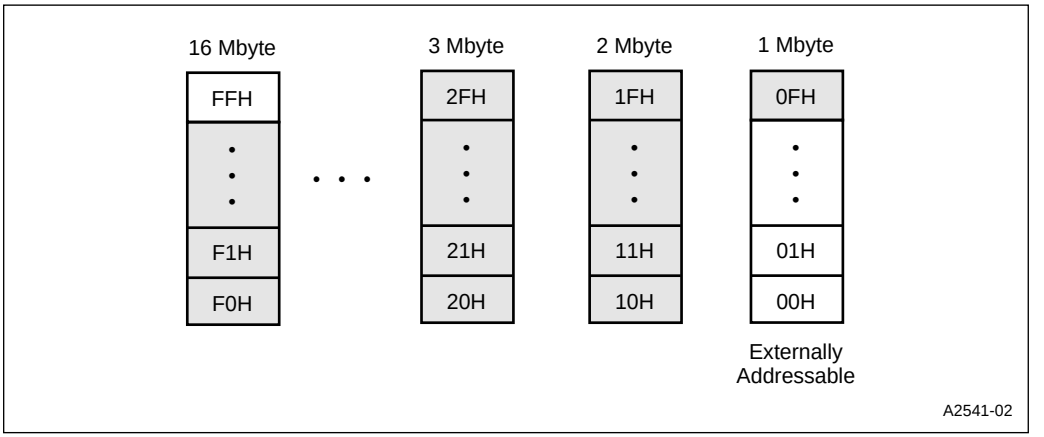


Figure 5-1. 16-Mbyte Address Space

The memory pages of interest are 00H–0EH and FFH. Pages 01H–0EH are external memory with unspecified contents; they can store either code or data. Pages 00H and FFH, shown in Figure 5-2, have special significance. Page 00H contains the register file and the special-function registers (SFRs), while page FFH contains special-purpose memory (chip configuration bytes and interrupt vectors) and program memory. The device fetches its first instruction from location FF2080H. Addresses in page FFH exist only in the internal 24-bit address space.

The implementation of page FFH in the 83C196NP differs from that in the 80C196NP and 80C196NU. For the 83C196NP, locations FF2000–FF2FFFH are implemented by 4 Kbytes of internal ROM and the remainder of page FFH (FF3000–FFFFFFH) is implemented by external memory in page 0FH. For the 80C196NP and the 80C196NU, which have no internal ROM, all of page FFH is implemented by external memory in page 0FH.

NOTE

Because the device has 24 bits of address internally, all programs must be written as though the device uses all 24 bits. The device resets from page FFH, so all code must originate from this page. (Use the assembler directive, "cseg at 0FF*xxxx*H.") This is true even if the code is actually stored in external memory.

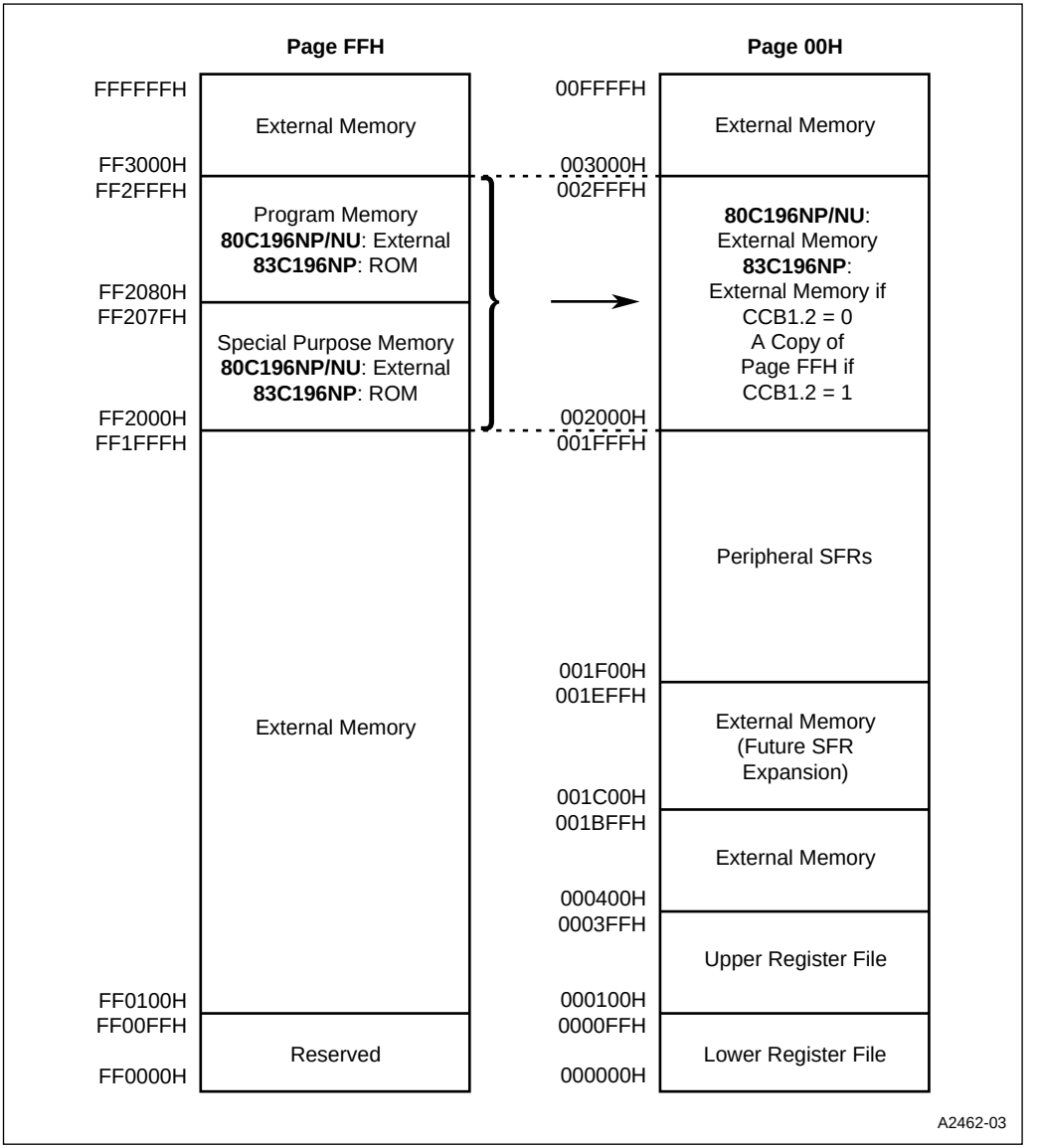


Figure 5-2. Pages FFH and 00H

5.2 MEMORY PARTITIONS

Table 5-1 is a memory map of the 8XC196NP and 80C196NU. The remainder of this section describes the partitions.

Table 5-1. 8XC196NP and 80C196NU Memory Map

Hex Address	Description	Addressing Modes
FFFFFF FF3000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
FF2FFF FF2080	Program memory (Note 1) After a device reset, the first instruction fetch is from FF2080H (or F2080H in external memory).	Indirect, indexed, extended
FF207F FF2000	Special-purpose memory (Note 1)	Indirect, indexed, extended
FF1FFF FF0100	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
FF00FF FF0000	Reserved (Note 2)	—
FEFFFF 0F0000	Overlaid memory; xF0000—xF00FFH are reserved	Indirect, indexed, extended
0EFFFF 010000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
00FFFF 003000	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
002FFF 002000	External device (memory or I/O) connected to address/data bus (Note 3)	Indirect, indexed, extended
001FFF 001F00	Peripheral SFRs (Note 4)	Indirect, indexed, extended, windowed direct
001EFF 001C00	External device (memory or I/O) connected to address/data bus; future SFR expansion (Note 5)	Indirect, indexed, extended
001BFF 000400	External device (memory or I/O) connected to address/data bus	Indirect, indexed, extended
0003FF 000100	Upper register file (register RAM)	Indirect, indexed, windowed direct
0000FF 000000	Lower register file (register RAM, stack pointer, CPU SFRs)	Direct, indirect, indexed

NOTES:
1. For the 80C196NP and 80C196NU, the program and special-purpose memory locations (FF2000–FF2FFFH) reside in external memory. For the 83C196NP, these locations can reside either in external memory or in internal ROM.
2. Do not use these locations except to initialize them. Except as otherwise noted, initialize unused program memory locations and reserved memory locations to FFH.
3. For the 80C196NP and 80C196NU, locations 002000–002FFFH reside in external memory. For the 83C196NP, locations 002000–002FFFH can be external memory (CCB1.2=0) or a copy of program and special-purpose memory stored in the internal ROM (CCB1.2=1).
4. For the 8XC196NP, locations 1FE0–1FFFH contain memory-mapped SFRs. They must be accessed with indirect, indexed, or extended addressing and they cannot be windowed.
5. **WARNING:** The contents or functions of these locations may change with future device revisions, in which case a program that relies on one or more of these locations might not function properly.

5.2.1 External Memory

Several partitions in pages 00H and FFH and all of pages 01H–0EH are assigned to external memory (see Table 5-1). Data can be stored in any part of this memory. Instructions can be stored in any part of this memory in 1-Mbyte mode, but can be stored only in page FFH in 64-Kbyte mode. “Memory Configuration Examples” on page 5-27 contains examples of memory configurations in the two modes. Chapter 13, “Interfacing with External Memory,” describes the external memory interface and shows additional examples of external memory configurations.

5.2.2 Program and Special-purpose Memory

Program memory and special-purpose memory occupy a 4-Kbyte memory partition from FF2000–FF2FFFH. For the 80C196NP and 80C196NU, this partition resides in external memory (external addresses F2000–F2FFFH). For the 83C196NP, this partition resides in on-chip ROM in page FFH, and it can also be mapped to page 00H (see “Remapping Internal ROM (83C196NP Only)” on page 5-22).

5.2.2.1 Program Memory in Page FFH

Three partitions in page FFH can be used for program memory:

- FF0100–FF1FFFH in external memory (external addresses F0100–F1FFFH)
- FF2080–FF2FFFH
  - — **80C196NP and 80C196NU:** This partition is in external memory (external addresses F2080–F2FFFH).
  - — **83C196NP:** The REMAP bit (CCB1.2), the EA# input, and the type of instruction (extended or nonextended) control access to this partition, as shown in Table 5-2.

Table 5-2. Program Memory Access for the 83C196NP

REMAP (CCB1.2)	EA# Pin	Instruction Type	Memory Location Accessed
X	Asserted	Extended or nonextended	External memory, F2080–F2FFFH
0	Deasserted	Extended or nonextended	Internal ROM, FF2080–FF2FFFH
1	Deasserted	Extended	Internal ROM, FF2080–FF2FFFH
		Nonextended	External memory, 02080–02FFFH

- FF3000–FFFFFH in external memory (external addresses F3000–FFFFFH)

NOTE

We recommend that you write FFH (the opcode for the RST instruction) to unused program memory locations. This causes a device reset if a program unintentionally begins to execute in unused memory.

5.2.2.2 Special-purpose Memory

Special-purpose memory resides in locations FF2000–FF207FH. It contains several reserved memory locations, the chip configuration bytes (CCBs), and vectors for both peripheral transaction server (PTS) and standard interrupts. Note that the special-purpose memory partition of the 80C196NU differs slightly from that of the 8XC196NP. Table 5-3 describes the special-purpose memory; bold type highlights the differences.

Table 5-3. 8XC196NP and 80C196NU Special-purpose Memory Addresses

8XC196NP Address (Hex)	80C196NU Address (Hex)	Description
FF207F **FF205E**	FF207F **FF2060**	Reserved (each byte must contain FFH)
FF205D FF2040	**FF205F** FF2040	PTS vectors
FF203F FF2030	FF203F FF2030	Upper interrupt vectors
FF202F FF201B	FF202F FF201B	Reserved (each byte must contain FFH)
FF201A	FF201A	CCB1
FF2019	FF2019	Reserved (must contain 20H)
FF2018	FF2018	CCB0
FF2017 **FF2014**	FF2017 **FF2010**	Reserved (each byte must contain FFH)
FF2013 FF2000	**FF200F** FF2000	Lower interrupt vectors

— **80C196NP and 80C196NU:** This partition is in external memory (external addresses F2000–F207FH).

— **83C196NP:** The REMAP bit (CCB1.2), the EA# input, and the type of instruction (extended or nonextended) control access to this partition, as shown in Table 5-4.

Table 5-4. Special-purpose Memory Access for the 83C196NP

REMAP (CCB1.2)	EA# Pin	Instruction Type	Memory Location Accessed
X	Asserted	Extended or nonextended	External memory, F2000–F207FH
0	Deasserted	Extended or nonextended	Internal ROM, FF2000–FF207FH
1	Deasserted	Extended	Internal ROM, FF2000–FF207FH
		Nonextended	External memory, 02000–0207FH

5.2.2.3 Reserved Memory Locations

Several memory locations are reserved for testing or for use in future products. Do not read or write these locations except to initialize them to the values shown in Table 5-3. The function or contents of these locations may change in future revisions; software that uses reserved locations may not function properly.

5.2.2.4 Interrupt and PTS Vectors

The peripheral transaction server (PTS) vectors contain the addresses of the PTS control blocks. The upper and lower interrupt vectors contain the addresses of the interrupt service routines. See Chapter 6, “Standard and PTS Interrupts,” for more information.

5.2.2.5 Chip Configuration Bytes

The chip configuration bytes (CCB0 and CCB1) specify the operating environment. They specify the bus width, bus mode (multiplexed or demultiplexed), write-control mode, wait states, power-down enabling, and the operating mode (1-Mbyte or 64-Kbyte mode). For the 83C196NP, CCB1 also controls ROM remapping. For the 80C196NP and 80C196NU, the CCBs are stored in external memory (locations F2018–F201AH). For the 83C196NP, the CCBs can be stored either in external memory (locations F2018–F201AH) or in the on-chip ROM (locations FF2018–FF201AH).

The chip configuration bytes are the first bytes fetched from memory when the device leaves the reset state. The post-reset sequence loads the CCBs into the chip configuration registers (CCRs). Once they are loaded, the CCRs cannot be changed until the next device reset. Typically, the CCBs are programmed once when the user program is compiled and are not redefined during normal operation. “Chip Configuration Registers and Chip Configuration Bytes” on page 13-14 describes the CCBs and CCRs.

5.2.3 Peripheral Special-function Registers (SFRs)

Locations 1F00–1FFFH provide access to the peripheral SFRs (see Table 5-5). Locations in this range that are omitted from the table are reserved. The peripheral SFRs are I/O control registers; they are physically located in the on-chip peripherals. Peripheral SFRs can be windowed and they can be addressed either as words or bytes, except as noted in the table.

Table 5-5. Peripheral SFRs

Reserved Locations		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FEEH	Reserved	Reserved
1FECH	Reserved	Reserved
1FEAH	Reserved	Reserved
1FE8H	Reserved	Reserved
Ports 1–4 SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FDEH	P4_PIN	P3_PIN
1FDCH	P4_REG	P3_REG
1FDAH	P4_DIR	P3_DIR
1FD8H	P4_MODE	P3_MODE
1FD6H	P2_PIN	P1_PIN
1FD4H	P2_REG	P1_REG
1FD2H	P2_DIR	P1_DIR
1FD0H	P2_MODE	P1_MODE
1FCEH	Reserved	Reserved
• • •	• • •	• • •
1FC0H	Reserved	Reserved
EPA, Timer 1, and Timer 2 SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F9EH	Reserved	EPA_PEND †††
1F9CH	Reserved	EPA_MASK
1F9AH	Reserved	Reserved
1F98H	Reserved	Reserved
†1F96H	TIMER2 (H)	TIMER2 (L)
1F94H	Reserved	T2CONTROL
†1F92H	TIMER1 (H)	TIMER1 (L)
1F90H	Reserved	T1CONTROL
†1F8EH	EPA3_TIME (H)	EPA3_TIME (L)
†1F8CH	EPA3_CON (H)	EPA3_CON (L)
†1F8AH	EPA2_TIME (H)	EPA2_TIME (L)
1F88H	Reserved	EPA2_CON
†1F86H	EPA1_TIME (H)	EPA1_TIME (L)
†1F84H	EPA1_CON (H)	EPA1_CON (L)
†1F82H	EPA0_TIME (H)	EPA0_TIME (L)
1F80H	Reserved	EPA0_CON

EPORT SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
††1FE6H	EP_PIN	Reserved
††1FE4H	EP_REG	Reserved
††1FE2H	EP_DIR	Reserved
††1FE0H	EP_MODE	Reserved
Serial I/O and PWM SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1FBEH	Reserved	Reserved
1FBCH	SP_BAUD (H)	SP_BAUD (L)
1FBAH	SP_CON	SBUF_TX
1FB8H	SP_STATUS	SBUF_RX
1FB6H	Reserved	CON_REG0
1FB4H	Reserved	PWM2_CONTROL
1FB2H	Reserved	PWM1_CONTROL
1FB0H	Reserved	PWM0_CONTROL
1FAEH	Reserved	Reserved
• • •	• • •	• • •
1FA0H	Reserved	Reserved
Chip-select SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
†1F6EH	Reserved	Reserved
†1F6CH	Reserved	BUSCON5
†1F6AH	ADDRMSK5 (H)	ADDRMSK5 (L)
1F68H	ADDRCOM5 (H)	ADDRCOM5 (L)
†1F66H	Reserved	Reserved
†1F64H	Reserved	BUSCON4
†1F62H	ADDRMSK4 (H)	ADDRMSK4 (L)
1F60H	ADDRCOM4 (H)	ADDRCOM4 (L)
1F5EH	Reserved	Reserved
1F5CH	Reserved	BUSCON3
1F5AH	ADDRMSK3 (H)	ADDRMSK3 (L)
1F58H	ADDRCOM3 (H)	ADDRCOM3 (L)
1F56H	Reserved	Reserved
1F54H	Reserved	BUSCON2
1F52H	ADDRMSK2 (H)	ADDRMSK2 (L)
1F50H	ADDRCOM2 (H)	ADDRCOM2 (L)

† Must be addressed as a word.

†† For the 8XC196NP, these are memory-mapped locations. They must be addressed with indirect or indexed instructions, and they cannot be windowed.

††† The EPA_PEND register was called EPA_STAT in previous documentation for the 8XC196NP.

†††† The 8XC196NP can be identified by its signature word, 80EFH, at locations 1F46–1F47H. The 8XC196NU has no signature word; locations 1F46–1F47H are reserved.

Table 5-5. Peripheral SFRs (Continued)

EPA, Timer 1, and Timer 2 SFRs (Continued)		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F7EH	Reserved	Reserved
1F7CH	Reserved	Reserved
1F7AH	Reserved	Reserved
1F78H	Reserved	Reserved
1F76H	Reserved	Reserved
1F74H	Reserved	Reserved
1F72H	Reserved	Reserved
1F70H	Reserved	Reserved

Chip-select SFRs (Continued)		
Address	**High (Odd) Byte**	**Low (Even) Byte**
1F4EH	Reserved	Reserved
1F4CH	Reserved	BUSCON1
1F4AH	ADDRMSK1 (H)	ADDRMSK1 (L)
1F48H	ADDRCOM1 (H)	ADDRCOM1 (L)
1F46H	Signature (H)††††	Signature (L)††††
1F44H	Reserved	BUSCON0
1F42H	ADDRMSK0 (H)	ADDRMSK0 (L)
1F40H	ADDRCOM0 (H)	ADDRCOM0 (L)

† Must be addressed as a word.
†† For the 8XC196NP, these are memory-mapped locations. They must be addressed with indirect or indexed instructions, and they cannot be windowed.
††† The EPA_PEND register was called EPA_STAT in previous documentation for the 8XC196NP.
†††† The 8XC196NP can be identified by its signature word, 80EFH, at locations 1F46–1F47H. The 8XC196NU has no signature word; locations 1F46–1F47H are reserved.

NOTE

Using any SFR as a base or index register for indirect or indexed operations can cause unpredictable results because external events can change the contents of SFRs. Also, because some SFRs are cleared when read, consider the implications of using an SFR as an operand in a read-modify-write instruction (e.g., XORB).

5.2.4 Register File

The register file is divided into an upper register file and a lower register file (Figure 5-3). The upper register file consists of general-purpose register RAM. The lower register file contains additional general-purpose register RAM along with the stack pointer (SP) and the CPU special-function registers (SFRs).

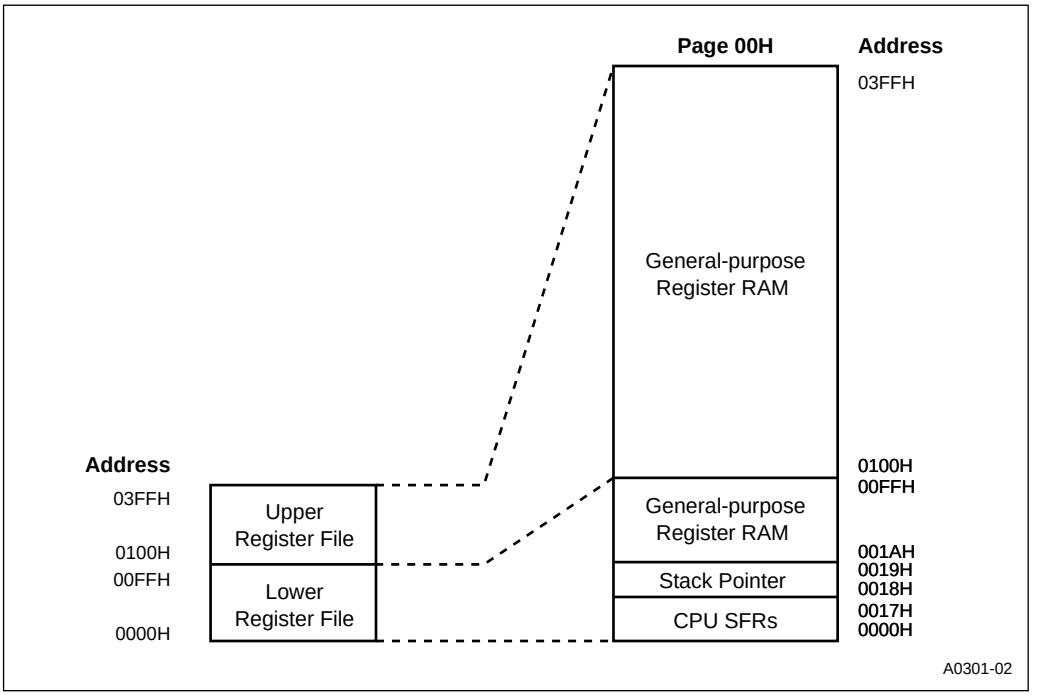


Figure 5-3. Register File Memory Map

Table 5-6 on page 5-11 lists the register file memory addresses. The RALU accesses the lower register file directly, without the use of the memory controller. It also accesses a *windowed* location directly (see "Windowing" on page 5-13). Only the upper register file and the peripheral SFRs can be windowed. Registers in the lower register file and registers being windowed can be accessed with direct addressing.

NOTE

The register file must not contain code. An attempt to execute an instruction from a location in the register file causes the memory controller to fetch the instruction from external memory.

Table 5-6. Register File Memory Addresses

Address Range	Description	Addressing Modes
03FFH 0100H	General-purpose register RAM; upper register file	Indirect, indexed, windowed direct
00FFH 001AH	General-purpose register RAM; lower register file	Direct, indirect, indexed
0019H 0018H	Stack pointer (SP); lower register file	Direct, indirect, indexed
0017H 0000H	CPU special-function registers (SFRs); lower register file	Direct, indirect, indexed

5.2.4.1 General-purpose Register RAM

The lower register file contains general-purpose register RAM. The stack pointer locations can also be used as general-purpose register RAM when stack operations are not being performed. The RALU can access this memory directly, using direct addressing.

The upper register file also contains general-purpose register RAM. The RALU normally uses indirect or indexed addressing to access the RAM in the upper register file. Windowing enables the RALU to use direct addressing to access this memory. (See Chapter 4, “Programming Considerations,” for a discussion of addressing modes.) Windowing provides fast context switching of interrupt tasks and faster program execution. (See “Windowing” on page 5-13.) PTS control blocks and the stack are most efficient when located in the upper register file.

5.2.4.2 Stack Pointer (SP)

Memory locations 0018H and 0019H contain the stack pointer (SP). The SP contains the address of the stack. The SP must point to a word (even) address that is two bytes (for 64-Kbyte mode) or four bytes (for 1-Mbyte mode) greater than the desired starting address. Before the CPU executes a subroutine call or interrupt service routine, it decrements the SP (by two in 64-Kbyte mode; by four in 1-Mbyte mode). Next, it copies (PUSHes) the address of the next instruction from the program counter onto the stack. It then loads the address of the subroutine or interrupt service routine into the program counter. When it executes the return-from-subroutine (RET) instruction at the end of the subroutine or interrupt service routine, the CPU loads (POPs) the contents of the top of the stack (that is, the return address) into the program counter. Finally, it increments the SP (by two in 64-Kbyte mode; by four in 1-Mbyte mode).

Subroutines may be nested. That is, each subroutine may call other subroutines. The CPU PUSHes the contents of the program counter onto the stack each time it executes a subroutine call. The stack grows downward as entries are added. The only limit to the nesting depth is the amount of available memory. As the CPU returns from each nested subroutine, it POPs the address off the top of the stack, and the next return address moves to the top of the stack.

Your program must load a word-aligned (even) address into the stack pointer. Select an address that is two bytes (for 64-Kbyte mode) or four bytes (for 1-Mbyte mode) greater than the desired starting address because the CPU automatically decrements the stack pointer before it pushes the first byte of the return address onto the stack. Remember that the stack grows downward, so allow sufficient room for the maximum number of stack entries. The stack must be located in page 00H, in either the internal register file or external RAM. The stack can be used most efficiently when it is located in the upper register file.

The following example initializes the top of the upper register file as the stack.

```
LD   SP, #400H              ;Load stack pointer
```

5.2.4.3 CPU Special-function Registers (SFRs)

Locations 0000–0017H in the lower register file are the CPU SFRs. Table 5-7 lists the CPU SFRs for the 8XC196NP and the 80C196NU and highlights those that are unique to the 80C196NU. Appendix C describes the CPU SFRs.

Table 5-7. CPU SFRs

8XC196NP CPU SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
0016H	Reserved	Reserved
0014H	Reserved	WSR
0012H	INT_MASK1	INT_PEND1
0010H	Reserved	Reserved
000EH	Reserved	Reserved
000CH	Reserved	Reserved
000AH	Reserved	Reserved
0008H	INT_PEND	INT_MASK
0006H	PTSSRV (H)	PTSSRV (L)
0004H	PTSSEL (H)	PTSSEL (L)
0002H	ONES_REG (H)	ONES_REG (L)
0000H	ZERO_REG (H)	ZERO_REG (L)

80C196NU CPU SFRs		
Address	**High (Odd) Byte**	**Low (Even) Byte**
0016H	Reserved	Reserved
0014H	WSR1†	WSR
0012H	INT_MASK1	INT_PEND1
0010H	Reserved	Reserved
000EH††	ACC_03†	ACC_02†
000CH††	ACC_01†	ACC_00†
000AH	ACC_STAT†	Reserved
0008H	INT_PEND	INT_MASK
0006H	PTSSRV (H)	PTSSRV (L)
0004H	PTSSEL (H)	PTSSEL (L)
0002H	ONES_REG (H)	ONES_REG (L)
0000H	ZERO_REG (H)	ZERO_REG (L)

† These SFRs are unique to the 80C196NU.
†† Must be addressed as a word.

5.3 WINDOWING

Windowing expands the amount of memory that is accessible with direct addressing. Direct addressing can access the lower register file with short, fast-executing instructions. With windowing, direct addressing can also access the upper register file and peripheral SFRs.

Windowing maps a segment of higher memory (the upper register file or peripheral SFRs) into the lower register file. The 8XC196NP has a single window selection register, while the 80C196NU has two. The first, WSR, is the same in both devices. WSR selects a 32-, 64-, or 128-byte segment of higher memory to be windowed into the top of the lower register file space.

The second, WSR1, is unique to the 80C196NU. WSR1 selects a 32- or 64-byte segment of higher memory to be windowed into the middle of the lower register file (Figure 5-4). Because the areas in the lower register file do not overlap, two windows can be in effect at the same time. For example, you can activate a 128-byte window using WSR and a 64-byte window using WSR1 (Figure 5-4). These two windows occupy locations 0040–00FFH in the lower register file, leaving locations 001A–003FH for use as general-purpose register RAM, locations 0018–0019H for the stack pointer or general-purpose register RAM, and locations 0000–0017H for the CPU SFRs.

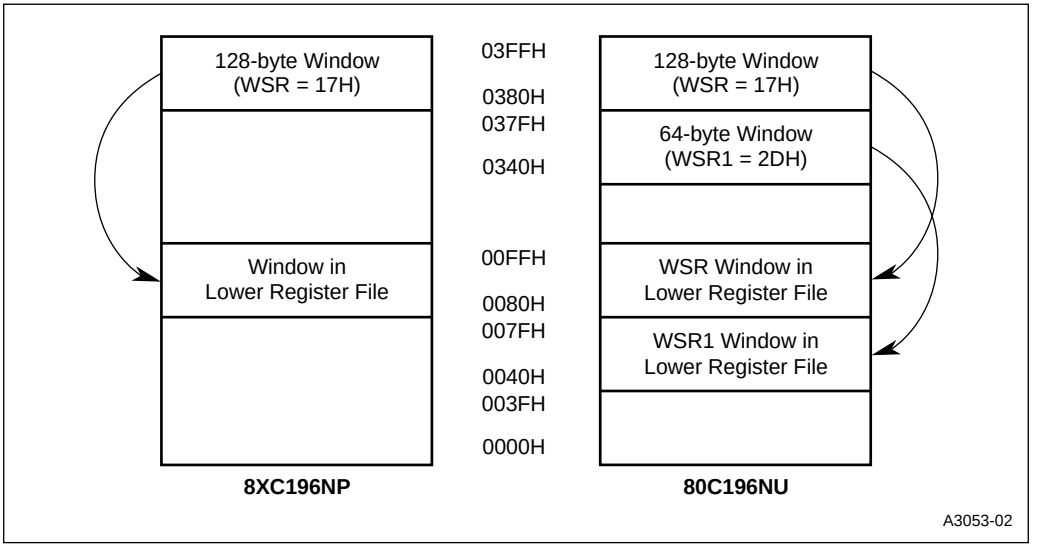


Figure 5-4. Windowing

5.3.1 Selecting a Window

The window selection register (Figure 5-5) has two functions. The HLDEN bit (WSR.7) enables and disables the bus-hold protocol (see Chapter 13, "Interfacing with External Memory"); it is unrelated to windowing. The remaining bits select a window to be mapped into the top of the lower register file. Window selection register 1 (Figure 5-6) selects a second window to be mapped into the middle of the 80C196NU's lower register file.

Table 5-8 provides a quick reference of WSR values for windowing the peripheral SFRs. Table 5-9 on page 5-15 lists the WSR values for windowing the upper register file.

WSR

Address: 0014H
Reset State: 00H

The window selection register (WSR) has two functions. One bit enables and disables the bus-hold protocol. The remaining bits select windows. Windows map sections of RAM into the top of the lower register file, in 32-, 64-, or 128-byte increments. PUSHA saves this register on the stack and POPA restores it.

7							0
HLDEN	W6	W5	W4	W3	W2	W1	W0

Bit Number	Bit Mnemonic	Function
7	HLDEN	HOLD#, HLDA# Protocol Enable This bit enables and disables the bus-hold protocol (see Chapter 13, "Interfacing with External Memory"). It has no effect on windowing. 1 = enable 0 = disable
6:0	W6:0	Window Selection These bits specify the window size and window number. See Table 5-8 on page 5-15 or Table 5-9 on page 5-15.

Figure 5-5. Window Selection (WSR) Register

WSR1
(80C196NU)

Address: 0015H
Reset State: 00H

Window selection 1 (WSR1) register selects a 32- or 64-byte segment of the upper register file or peripheral SFRs to be windowed into the middle of the lower register file, below any window selected by the WSR.

	7							0
80C196NU	—	W6	W5	W4	W3	W2	W1	W0

Bit Number	Bit Mnemonic	Function
7	—	Reserved; always write as zero.
6:0	W6:0	Window Selection These bits specify the window size and window number. See Table 5-8 on page 5-15 or Table 5-9 on page 5-15.

Figure 5-6. Window Selection 1 (WSR1) Register

Table 5-8. Selecting a Window of Peripheral SFRs

Peripheral	WSR or WSR1 Value for 32-byte Window (00E0–00FFH or 0060–007FH)	WSR or WSR1 Value for 64-byte Window (00C0–00FFH or 0040–007FH)	WSR Value for 128-byte Window (0080–00FFH)
EPORT†	7FH†		
Ports 1–4	7EH	3FH†	
PWM and SIO	7DH		
EPA and Timers	7CH	3EH	1FH†
Chip selects 4–5	7BH		
Chip selects 0–3	7AH	3DH	1EH

† For the 8XC196NP, the EPORT SFRs are memory-mapped SFRs. They must be accessed with indirect, indexed, or extended addressing; they cannot be windowed.

Table 5-9. Selecting a Window of the Upper Register File

Register RAM Locations (Hex)	WSR or WSR1 Value for 32-byte Window (00E0–00FFH or 0060–007FH)	WSR or WSR1 Value for 64-byte Window (00C0–00FFH or 0040–007FH)	WSR Value for 128-byte Window (0080–00FFH)
03E0–03FF	5FH		
03C0–03DF	5EH	2FH	
03A0–03BF	5DH		
0380–039F	5CH	2EH	17H

Table 5-9. Selecting a Window of the Upper Register File (Continued)

Register RAM Locations (Hex)	WSR or WSR1 Value for 32-byte Window (00E0–00FFH or 0060–007FH)	WSR or WSR1 Value for 64-byte Window (00C0–00FFH or 0040–007FH)	WSR Value for 128-byte Window (0080–00FFH)
0360–037F	5BH		
0340–035F	5AH	2DH	
0320–033F	59H		
0300–031F	58H	2CH	16H
02E0–02FF	57H		
02C0–02DF	56H	2BH	
02A0–02BF	55H		
0280–029F	54H	2AH	15H
0260–027F	53H		
0240–025F	52H	29H	
0220–023F	51H		
0200–021F	50H	28H	14H
01E0–01FF	4FH		
01C0–01DF	4EH	27H	
01A0–01BF	4DH		
0180–019F	4CH	26H	13H
0160–017F	4BH		
0140–015F	4AH	25H	
0120–013F	49H		
0100–011F	48H	24H	12H

5.3.2 Addressing a Location Through a Window

After you have selected the desired window, you need to know the direct address of the memory location (the address in the lower register file). For SFRs, refer to the WSR tables in Appendix C. For register file locations, calculate the direct address as follows:

1. Subtract the base address of the area to be remapped (from Table 5-10 on page 5-17) from the address of the desired location. This gives you the offset of that particular location.
2. Add the offset to the base address of the window (from Table 5-11). The result is the direct address.

Table 5-10. Windows

Base Address (Hex)	WSR or WSR1 Value for 32-byte Window (00E0–00FFH or 0060–007FH)	WSR or WSR1 Value for 64-byte Window (00C0–00FFH or 0040–007FH)	WSR Value for 128-byte Window (0080–00FFH)
Peripheral SFRs			
†1FE0	†7FH		
1FC0	7EH	†3FH	
1FA0	7DH		
1F80	7CH	3EH	†1FH
1F60	7BH		
1F40	7AH	3DH	
1F20	79H		
1F00	78H	3CH	1EH
Upper Register File			
03E0H	5FH		
03C0H	5EH	2FH	
03A0H	5DH		
0380H	5CH	2EH	17H
0360H	5BH		
0340H	5AH	2DH	
0320H	59H		
0300H	58H	2CH	16H
02E0H	57H		
02C0H	56H	2BH	
02A0H	55H		
0280H	54H	2AH	15H
0260H	53H		
0240H	52H	29H	
0220H	51H		
0200H	50H	28H	14H
01E0H	4FH		
01C0H	4EH	27H	
01A0H	4DH		
0180H	4CH	26H	13H
0160H	4BH		
0140H	4AH	25H	
0120H	49H		
0100H	48H	24H	12H

† For the 8XC196NP, locations 1FE0–1FFFH contain memory-mapped SFRs that cannot be windowed. Reading these locations through a window returns FFH; writing these locations through a window has no effect. For the 80C196NU, these locations are **not** memory-mapped; they **can** be windowed.

Table 5-11. Windowed Base Addresses

Window Size	WSR Windowed Base Address (Base Address in Lower Register File)	WSR1 Windowed Base Address (Base Address in Lower Register File) 80C196NU Only
32-byte	00E0H	0060H
64-byte	00C0H	0040H
128-byte	0080H	—

Appendix C includes a table of the windowable SFRs with the window selection register values and direct addresses for each window size. The following examples explain how to determine the WSR value and direct address for any windowable location. An additional example shows how to set up a window by using the linker locator.

5.3.2.1 32-byte Windowing Example

Assume that you wish to access location 014BH (a location in the upper register file used for general-purpose register RAM) with direct addressing through a 32-byte window. Table 5-10 on page 5-17 shows that you need to write 4AH to the window selection register. It also shows that the base address of the 32-byte memory area is 0140H. To determine the offset, subtract that base address from the address to be accessed (014BH – 0140H = 000BH). Add the offset to the base address of the window in the lower register file (from Table 5-11). The direct address is 00EBH (000BH + 00E0H) for a WSR window or 006BH (000BH + 0060H) for a WSR1 window.

5.3.2.2 64-byte Windowing Example

Assume that you wish to access the SFR at location 1F8CH with direct addressing through a 64-byte window. Table 5-10 on page 5-17 shows that you need to write 3EH to the window selection register. It also shows that the base address of the 64-byte memory area is 1F80H. To determine the offset, subtract that base address from the address to be accessed (1F8CH – 1F80H = 000CH). Add the offset to the base address of the window in the lower register file (from Table 5-11). The direct address is 00CCH (000CH + 00C0H) for a WSR window or 004CH (000CH + 0040H) for a WSR1 window.

5.3.2.3 128-byte Windowing Example

Assume that you wish to access the SFR at location 1F82H with direct addressing through a 128-byte window. Table 5-11 on page 5-18 shows that you need to write 1FH to the window selection register. It also shows that the base address of the 128-byte memory area is 1F80H. To determine the offset, subtract that base address from the address to be accessed (1F82H – 1F80H = 0002H). Add the offset to the base address of the window in the lower register file (from Table 5-11). The direct address is 0082H (0002H + 0080H).

5.3.2.4 Unsupported Locations Windowing Example (8XC196NP Only)

Assume that you wish to access location 1FE7H (the EP_PIN register, a memory-mapped SFR) with direct addressing through a 128-byte window. This location is in the range of addresses (1FE0–1FFFH) that cannot be windowed. Although you could set up the window by writing 1FH to the WSR, reading this location through the window would return FFH (all ones) and writing to it would not change the contents. However, you could directly address the remaining SFRs in the range of 1F80–1FDFH.

5.3.2.5 Using the Linker Locator to Set Up a Window

In this example, the linker locator is used to set up a window. The linker locator locates the window in the upper register file and determines the value to load in the WSR for access to that window. (Please consult the manual provided with the linker locator for details.)

```
*********  mod1  **************
mod1 module main              ;Main module for linker
public function1
extrn  ?WSR                   ;Must declare ?WSR as external

wsr  equ   14h:byte
sp   equ   18h:word

oseg
     var1:   dsw  1            ;Allocate variables in an overlayable segment
     var2:   dsw  1
     var3:   dsw  1

cseg

function1:
     push  wsr                ;Prolog code for wsr
     ldb   wsr, #?WSR         ;Prolog code for wsr

     add var1, var2, var3     ;Use the variables as registers
     ;
     ;
     ;

     ldb  wsr, [sp]           ;Epilog code for wsr
     add sp, #2               ;Epilog code for wsr
     ret
end

********  mod2  **************
```

```
public function2
extrn  ?WSR

wsr  equ   14h:byte
sp   equ   18h:word

oseg
     var1:   dsw  1
     var2:   dsw  1
     var3:   dsw  1

cseg

function2:
     push  wsr               ;Prolog code for wsr
     ldb   wsr, #?WSR        ;Prolog code for wsr

     add var1, var2, var3
     ;
     ;
     ;

     ldb  wsr, [sp]          ;Epilog code for wsr
     add sp, #2              ;Epilog code for wsr
     ret
end
*****************************
```

The following is an example of a linker invocation to link and locate the modules and to determine the proper windowing.

```
RL196 MOD1.OBJ, MOD2.OBJ registers(100h-03ffh) windowsize(32)
```

The above linker controls tell the linker to use registers 0100–03FFH for windowing and to use a window size of 32 bytes. (These two controls enable windowing.)

The following is the map listing for the resultant output module (MOD1 by default):

```
SEGMENT MAP FOR mod1(MOD1):

            TYPE          BASE    LENGTH   ALIGNMENT    MODULE NAME
            ----          ----    ------   ---------    -----------
**RESERVED*               0000H     001AH
            STACK         001AH     0006H   WORD
*** GAP ***               0020H     00E0H
            OVRLY         0100H     0006H   WORD         MOD2
            OVRLY         0106H     0006H   WORD         MOD1
*** GAP ***               010CH     1F74H
            CODE          2080H     0011H   BYTE         MOD2
            CODE          2091H     0011H   BYTE         MOD1
*** GAP ***               20A2H     DF5EH
```

This listing shows the disassembled code:

```
2080H         ;C814               | PUSH  WSR
2082H         ;B14814             | LDB   WSR,#48H
2085H         ;44E4E2E0           | ADD   E0H,E2H,E4H
2089H         ;B21814             | LDB   WSR,[SP]
208CH         ;65020018           | ADD   SP,#02H
2090H         ;F0                 | RET
2091H         ;C814               | PUSH  WSR
2093H         ;B14814             | LDB   WSR,#48H
2096H         ;44EAE8E6           | ADD   E6H,E8H,EAH
209AH         ;B21814             | LDB   WSR,[SP]
209DH         ;65020018           | ADD   SP,#02H
20A1H         ;F0                 | RET
```

The C compiler can also take advantage of this feature if the “windows” switch is enabled. For details, see the MCS 96 microcontroller architecture software products in the *Development Tools Handbook*.

5.3.3 Windowing and Addressing Modes

Once windowing is enabled, the windowed locations can be accessed both through the window using direct addressing and through its actual address using indirect or indexed addressing. The lower register file locations that are covered by the window are always accessible by indirect or indexed operations. To re-enable direct access to the entire lower register file, clear bits 6:0 of the window selection register. To enable direct access to a particular location in the lower register file, you may select a smaller window that does not cover that location.

When windowing is enabled:

- a direct instruction that uses an address within the lower register file actually accesses the window in the upper register file;
- an indirect or indexed instruction that uses an address within either the lower register file or the upper register file accesses the actual location in memory.

The following sample code illustrates the difference between direct and indexed addressing when using windowing.

```
PUSHA                   ; Pushes the contents of WSR onto the stack
LDB  WSR, #17H          ; Selects window 17H, a 128-byte block
                        ; (windows 0380-03FFH into 0080-00FFH)
                        ; The next instruction uses direct addr
ADD  40H, 80H           ; mem_word(40H)←mem_word(40H) + mem_word(380H)
                        ; The next two instructions use indirect addr
ADD  40H, 80H[0]        ; mem_word(40H)←mem_word(40H) + mem_word(80H +0)
ADD  40H, 380H[0]       ; mem_word(40H)←mem_word(40H) + mem_word(380H +0)
POPA                    ; reloads the previous contents into WSR
```

5.4 REMAPPING INTERNAL ROM (83C196NP ONLY)

The 83C196NP's 4 Kbytes of ROM are located in FF2000–FF2FFFH. By using the REMAP bit (CCB1.2) and the EA# input, you can also access these locations in external memory (page 0FH or page 00H). The REMAP bit is loaded from CCB1 upon leaving reset and cannot be changed until the next reset. Tie EA# low to access external memory or tie it high to access the on-chip ROM. (Refer to the EA# description in Appendix B for additional information on using the EA# pin.)

NOTE

The EA# input is effective only for accesses to the 83C196NP's on-chip ROM (FF2000–FF2FFFH). For an access to any other location, the value of EA# is irrelevant.

Without remapping (CCB1.2 = 0), an access to FF2000–FF2FFFH is directed to internal ROM (FF2000–FF2FFFH) if EA# is high and to external memory (F2000–F2FFFH) if EA# is low. In either case, data in this area must be accessed with extended instructions.

With remapping enabled (CCB1.2 = 1) and EA# high, you can access the contents of FF2000–FF2FFFH in two ways:

- in internal ROM (FF2000–FF2FFFH) using an extended instruction
- in external memory (002000–002FFFH) using a nonextended instruction. This makes the far data in FF2000–FF2FFFH accessible as near data.

With remapping enabled (CCB1.2 = 1) and EA# low, you can access the contents of FF2000–FF2FFFH in external memory (F2000–F2FFFH) using an extended instruction.

An advantage of remapping ROM is that it makes the data in ROM accessible as near data in external memory page 00H. The data can then be accessed more quickly with nonextended instructions. An advantage of not remapping ROM is that the corresponding area in external memory page 00H is available for storing additional near data.

5.5 FETCHING CODE AND DATA IN THE 1-MBYTE AND 64-KBYTE MODES

This section describes how the device fetches instructions and accesses data in the 1-Mbyte and 64-Kbyte modes. When the device leaves reset, the MODE64 bit (CCB1.1) selects the 1-Mbyte or 64-Kbyte mode. The mode cannot be changed until the next reset.

NOTE

The 8XC196NP and 80C196NU have two major differences concerning code and data fetches. The 8XC196NP's prefetch queue is four bytes, while the 80C196NU's is eight bytes. The 8XC196NP gives higher priority to instruction fetches than to data fetches, while the 80C196NU gives higher priority to data accesses than to instruction fetches.

5.5.1 Fetching Instructions

The 24-bit program counter (Figure 5-7) consists of the 8-bit extended program counter (EPC) concatenated with the 16-bit master program counter (PC). It holds the address of the next instruction to be fetched. The page number of the instruction is in the EPC. In 1-Mbyte mode, the EPC can have any 8-bit value. However, only the four LSBs of the EPC are implemented externally, as EPORT pins A19:16. This means that in the 1-Mbyte mode, the device can fetch code from any page in the 1-Mbyte address space: 00H–0FH and FFH (FFH overlays 0FH). In 64-Kbyte mode, the EPC is fixed at FFH, which limits program memory to page FFH (and 0FH).

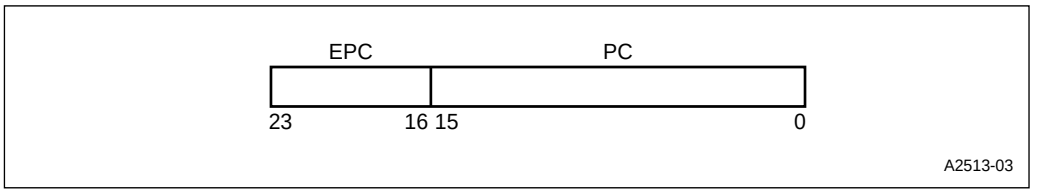


Figure 5-7. The 24-bit Program Counter

5.5.2 Accessing Data

Internally, data addresses have 24 bits (Figure 5-8 on page 5-24). The lower 16 bits are supplied by the 16-bit data address register. The upper 8 bits (the page number) come from different sources for nonextended and extended instructions. ("EPORT Operation" on page 7-12 describes how the page number is output to the EPORT pins.)

For nonextended instructions, the EP_REG register provides the page number. Data and constants in this page are called *near data* and *near constants*.

NOTE

The 8XC196NP allows you to change the value of EP_REG to control which memory page a nonextended instruction accesses. However, software tools require that EP_REG be equal to 00H. The 80C196NU forces all nonextended data accesses to page 00H. You cannot use EP_REG to change pages.

Data outside the page specified by EP_REG is called *far data*. To access far data, you must use extended instructions. For extended instructions, the CPU provides the page number.

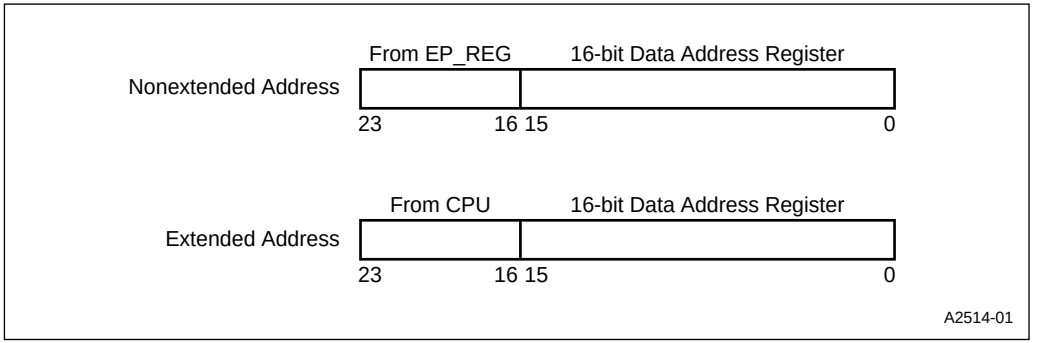


Figure 5-8. Formation of Extended and Nonextended Addresses

The code example below illustrates the use of extended instructions to access data in page 01H.

```
EP_REG          EQU 1FE5H
                RSEG AT 1CH
     TEMP:      DSW 1
     RESULT:    DSW 1
                CSEG AT 0FF2080H
                .                       ;some code
                .                       ;
                .                       ;
SUBB:           PUSHA                   ;save flags, disable interrupts
                LD   TEMP,#1234H        ;
                EST  TEMP,010600H       ;store temp value in 010600H
                ADD  RESULT,TEMP,#4000H ;do something with registers
                EST  RESULT,010602H     ;store result in 010602H
                .                       ;more eld/est instructions
                .                       ;
                .                       ;
                POPA                    ;restore flags and interrupts
                RET                     ;
                .                       ;more code
                .                       ;
                .                       ;
DONE:           BR DONE
                END
```

5.5.3 Code Fetches in the 1-Mbyte Mode

CCR1.1 (the MODE64 bit) controls whether the device operates in 1-Mbyte or 64-Kbyte mode. CCR1 is loaded with the contents of CCB1 at reset. When MODE64 is clear, the device operates in 1-Mbyte mode. In this mode, code can execute from any page in the 1-Mbyte address space. An extended jump, branch, or call instruction across pages changes the EPC value to the destination page. For example, assume that code is executing from page FFH. The following code segment branches to an external memory location in page 00H and continues execution.

```
0FF2090H:       LD   TEMP,#12H          ; code executing in page FFH
                ST   TEMP,PORT1         ; code executing in page FFH
                EBR  003000H            ; jump to location 3000H in page 00H
003000H:        ADD  TEMP,#50H          ; code executing in page 00H
```

Code fetches are from external memory or internal memory, depending on the device, the instruction address, and the value of the EA# input.

80C196NU:

Code executes from any page in external memory.

80C196NP:

For devices without internal nonvolatile memory, EA# must be tied low, and code executes from any page in external memory.

83C196NP:

Code in all locations except FF2000–FF2FFFH executes from external memory.

Instruction fetches from FF2000–FF2FFFH are controlled by the EA# input:

- If EA# is low, code executes from external memory.
- If EA# is high, code executes from internal ROM.

Note that the EA# input functions only for the address range FF2000–FF2FFFH.

5.5.4 Code Fetches in the 64-Kbyte Mode

CCR1.1 (the MODE64 bit) controls whether the device operates in 1-Mbyte or 64-Kbyte mode. CCR1 is loaded with the contents of CCB1 at reset. When MODE64 is set, the device operates in 64-Kbyte mode. In this mode, the EPC (Figure 5-7 on page 5-23) is fixed at FFH, which allows instructions to execute from page FFH only. Extended jump, branch, and call instructions do **not** function in the 64-Kbyte mode.

Code fetches are from external memory or internal memory, depending on the device, the memory location, and the value of the EA# input.

80C196NU:

Code executes from page 0FH in external memory. (The 80C196NU has no EA# input.)

80C196NP:

For devices without internal nonvolatile memory, EA# must be tied low, and code executes only from page 0FH in external memory.

83C196NP:

Code in all locations except FF2000–FF2FFFH executes from external memory.

Instruction fetches from FF2000–FF2FFFH are controlled by the EA# input:

- If EA# is low, code executes from external memory (page 0FH).
- If EA# is high, code executes from internal ROM (page FFH).

5.5.5 Data Fetches in the 1-Mbyte and 64-Kbyte Modes

Data fetches are the same in the 1-Mbyte and 64-Kbyte modes. The device can access data in any page. Data accesses to page 00H are nonextended. Data accesses to any other page are extended.

NOTE

This information on data fetches applies only for EP_REG = 00H.

80C196NP and 80C196NU:

Data accesses to the register file (0000–03FFH) and the SFRs (1F00–1FFFH) are directed to the internal registers. All other data accesses are directed to external memory.

83C196NP:

Data accesses to the register file (0000–03FFH) and the SFRs (1F00–1FFFH) are directed to the internal registers. Accesses to other locations are directed to external memory, except as noted below:

Data accesses to FF2000–FF2FFFH depend on the EA# input:

- If EA# is low, accesses are to external memory (page 0FH).
- If EA# is high, accesses are to the internal ROM (page FFH).

Data accesses to 002000–002FFFH depend on the REMAP bit and the EA# input:

- If remapping is disabled (CCB1.2 = 0), accesses are external.
- If remapping is enabled (CCB1.2 = 1), accesses depend on EA#:
  - — If EA# is low, accesses are external (REMAP is ignored).
  - — If EA# is high, accesses are to the internal ROM.

5.6 MEMORY CONFIGURATION EXAMPLES

This section provides examples of memory configurations for both 64-Kbyte and 1-Mbyte mode. Each example consists of a circuit diagram and a memory map that describes how the address space is implemented. Chapter 13, “Interfacing with External Memory,” discusses the interface in detail and provides additional examples.

5.6.1 Example 1: Using the 64-Kbyte Mode

Figure 5-9 shows a system designed for operation in the 64-Kbyte mode. Code executes only from page FFH, which is implemented by the 64-Kbyte flash memory. The 32-Kbyte RAM in the upper half of page 00H stores near data. Table 5-12 on page 5-28 lists the memory addresses for this example. (For memory map details, see Table 5-1 on page 5-4.)

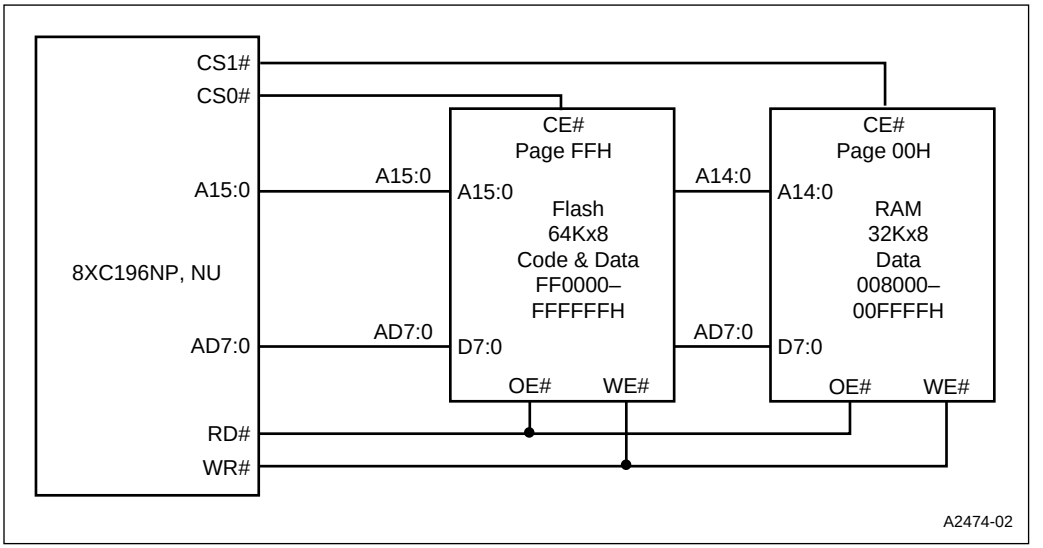


Figure 5-9. A 64-Kbyte System With an 8-bit Bus

80C196NP and 80C196NU: The flash memory, which implements page FFH, holds the special-purpose memory (FF2000–FF207FH), code, and far constants.

83C196NP only: Locations FF2000–FF2FFFH, which store code and special-purpose memory, are implemented by internal ROM. Data accesses to locations FF2000–FF2FFFH are directed to the flash memory if EA# is low and to internal ROM if EA# is high. Locations FF2000–FF2FFFH can be remapped to page 00H by setting the REMAP bit (CCB1.2). An access to the remapped area, 002000–002FFFH, is directed to ROM if EA# is high and to external memory if EA# is low. With remapping enabled (REMAP = 1) and EA# high, the far constants in the special-purpose memory can be accessed as near constants in page 00H.

Table 5-12. Memory Map for the System in Figure 5-9

Address	Description
FFFFFFH FF3000H	External flash memory (code or far constants)
FF2FFFH FF2080H	Program memory: **80C196NP and 80C196NU:** External flash memory **83C196NP**: Internal ROM (EA# = 1), external memory (EA# = 0)
FF207FH FF2000H	Special-purpose memory (far constants): **80C196NP and 80C196NU**: External flash memory **83C196NP**: Internal ROM (EA# = 1), external memory (EA# = 0)
FF1FFFH FF0100H	External flash memory (code or far constants)
FF00FFH FF0000H	Reserved
FEFFFFH 010000H	Unimplemented
00FFFFH 008000H	32-Kbyte external RAM (near data)
007FFFH 003000H	Unimplemented
002FFFH 002000H	**80C196NP and 80C196NU:** Unimplemented **83C196NP:** Program and special-purpose memory remapped from internal ROM (REMAP = 1; EA# = 1)
001FFFH 001F00H	Internal peripheral special-function registers (SFRs)
001EFFH 001C00H	Unimplemented (future SFR expansion)
001BFFH 000400H	Unimplemented
0003FFH 000100H	Upper register file (general-purpose register RAM)
0000FFH 000018H	Lower register file (general-purpose register RAM and stack pointer)
000017H 000000H	Lower register file (CPU SFRs)

5.6.2 Example 2: A 64-Kbyte System with Additional Data Storage

Figure 5-10 shows another system designed for operation in the 64-Kbyte mode. Code executes from page FFH only. This system is the same as the example in "Example 1: Using the 64-Kbyte Mode" on page 5-27, but with additional RAM. The 64-Kbyte RAM stores near data in page 00H. The 128-Kbyte RAM stores far data in pages 01H and 02H. Table 5-13 lists the memory addresses. (For memory map details, see Table 5-1 on page 5-4.)

80C196NP and 80C196NU: The flash memory, which implements page FFH, holds the special-purpose memory (FF2000–FF207FH), code, and far constants.

83C196NP only: Locations FF2000–FF2FFFH, which store code and special-purpose memory, are implemented by internal ROM. Data accesses to locations FF2000–FF2FFFH are directed to the flash memory if EA# is low and to internal ROM if EA# is high. Locations FF2000–FF2FFFH can be remapped to page 00H by setting the REMAP bit (CCB1.2). An access to the remapped area, 002000–002FFFH, is directed to ROM if EA# is high and to external memory if EA# is low. With remapping enabled (REMAP = 1) and EA# high, the far constants in the special-purpose memory can be accessed as near constants in page 00H.

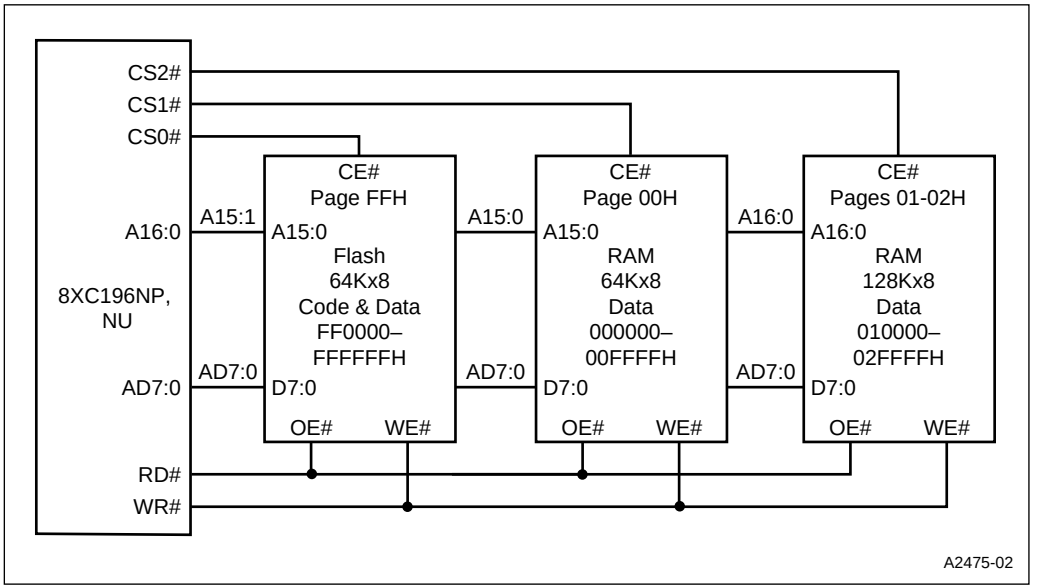


Figure 5-10. A 64-Kbyte System with Additional Data Storage

Table 5-13. Memory Map for the System in Figure 5-10

Address	Description
FFFFFFH FF3000H	External flash memory (code or far constants)
FF2FFFH FF2080H	Program memory: **80C196NP and 80C196NU**: External flash memory **83C196NP**: Internal ROM (EA# = 1), external memory (EA# = 0)
FF207FH FF2000H	Special-purpose memory: **80C196NP and 80C196NU**: External flash memory (far constants) **83C196NP**: Internal ROM (EA# = 1), external memory (EA# = 0)
FF1FFFH FF0100H	External flash memory (code or far constants)
FF00FFH FF0000H	Reserved
FEFFFFH 030000H	Unimplemented
02FFFFH 010000H	128-Kbyte external RAM (far data)
00FFFFH 003000H	External RAM (near data)
002FFFH 002000H	**80C196NP and 80C196NU:** External RAM **83C196NP:** External RAM (CCB1.2 = 0) or remapped internal ROM (CCB1.2 = 1)
001FFFH 001F00H	Internal peripheral special-function registers (SFRs)
001EFFH 001C00H	External RAM (future SFR expansion)
001BFFH 000400H	External RAM (near data)
0003FFH 000100H	Upper register file (general-purpose register RAM)
0000FFH 000018H	Lower register file (general-purpose register RAM and stack pointer)
000017H 000000H	Lower register file (CPU SFRs)

5.6.3 Example 3: Using 1-Mbyte Mode

Figure 5-11 shows a system designed for operation in the 1-Mbyte mode. In this mode, code can execute from any page in the 1-Mbyte memory space. The system uses both 8-bit and 16-bit buses and uses the write-strobe mode. (See Chapter 13, “Interfacing with External Memory.”)

The 32K×8 RAM stores near data in the upper half of page 00H. The 32K×16 RAM stores far data in page 01H. Using the WRL# and WRH# signals makes this RAM both byte- and word-accessible. The 128K×16 flash memory stores code and far constants in pages FCH, FDH, FEH, and FFH. With the write-signals connected as shown, the flash memory is word-accessible only. Table 5-14 lists the memory addresses. (For memory map details, see Table 5-3 on page 5-6.)

83C196NP only. The code and data in FF2000–FF2FFFH are implemented by internal ROM. Remapping this area into page 00H by setting the REMAP bit (CCB1.2) makes the far constants in FF2000–FF2FFFH of ROM accessible as near constants. An access to this address range is directed to external memory if EA# is low and to internal ROM if EA# is high.

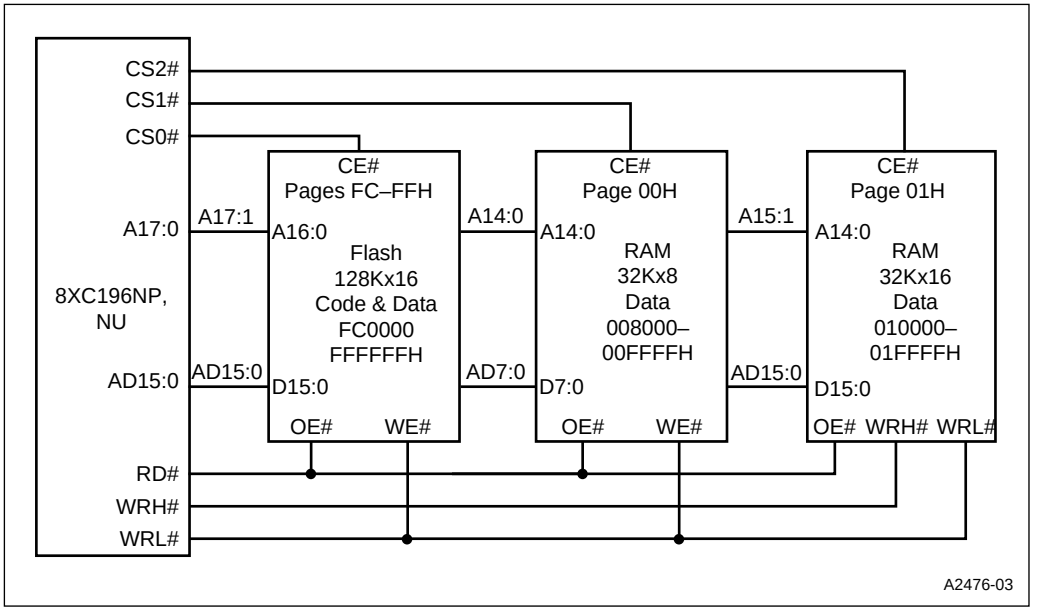


Figure 5-11. Example System Using the 1-Mbyte Mode

Notice that the microcontroller’s A1 line connects to a word-wide memory device’s A0 line. For a byte-wide memory, the microcontroller’s A0 line selects the byte to be read. For a word-wide memory, the microcontroller reads an entire word, then selects the required byte internally.

Table 5-14. Memory Map for the System in Figure 5-11

Address	Description
FFFFFFH FF3000H	External memory (code or far constants)
FF2FFFH FF2080H	Program memory: **80C196NP and 80C196NU**: External memory **83C196NP**: Internal ROM (EA# = 1), external memory (EA# = 0)
FF207FH FF2000H	Special-purpose memory: **80C196NP and 80C196NU**: external memory (far constants) **83C196NP**: Internal ROM (EA# = 1), external memory (EA# = 0)
FF1FFFH FF0100H	External flash memory (code or far constants)
FF00FFH FF0000H	Reserved
FEFFFFH FC0000H	External flash memory (far code, far constants)
FBFFFFH 020000H	Unimplemented
01FFFFH 010000H	64-Kbyte external RAM (far data)
00FFFFH 008000H	32-Kbyte external RAM (near data)
007FFFH 003000H	Unimplemented
002FFFH 002080H	**80C196NP and 80C196NU:** Unimplemented **83C196NP:** Program memory remapped from internal ROM (CCB1.2 = 1; EA# = 1)
00207FH 002000H	**80C196NP and 80C196NU**: Unimplemented **83C196NP:** Special-purpose memory (near constants) remapped from internal ROM (CCB1.2 = 1; EA# = 1)
001FFFH 001F00H	Internal peripheral special-function registers (SFRs)
001EFFH 001C00H	Unimplemented (future SFR expansion)
001BFFH 000400H	Unimplemented
0003FFH 000100H	Upper register file (general-purpose register RAM)
0000FFH 000018H	Lower register file (general-purpose register RAM and stack pointer)
000017H 000000H	Lower register file (CPU SFRs)

6

Standard and PTS Interrupts

CHAPTER 6
STANDARD AND PTS INTERRUPTS

This chapter describes the interrupt control circuitry, priority scheme, and timing for standard and peripheral transaction server (PTS) interrupts. It discusses the three special interrupts and the four PTS modes, two of which are used with the EPA to produce pulse-width modulated (PWM) outputs. It also explains interrupt programming and control.

6.1 OVERVIEW OF INTERRUPTS

The interrupt control circuitry within a microcontroller permits real-time events to control program flow. When an event generates an interrupt, the device suspends the execution of current instructions while it performs some service in response to the interrupt. When the interrupt is serviced, program execution resumes at the point where the interrupt occurred. An internal peripheral, an external signal, or an instruction can generate an interrupt request. In the simplest case, the device receives the request, performs the service, and returns to the task that was interrupted.

This microcontroller's flexible interrupt-handling system has two main components: the programmable interrupt controller and the peripheral transaction server (PTS). The programmable interrupt controller has a hardware priority scheme that can be modified by your software. Interrupts that go through the interrupt controller are serviced by interrupt service routines that you provide. The upper and lower interrupt vectors in special-purpose memory (see Chapter 5, "Memory Partitions") contain the lower 16 bits of the interrupt service routines' addresses. The CPU automatically adds FF0000H to the 16-bit vector in special-purpose memory to calculate the address of the interrupt service routine, and then executes the routine. The peripheral transaction server (PTS), a microcoded hardware interrupt processor, provides high-speed, low-overhead interrupt handling; it does not modify the stack or the PSW. You can configure most interrupts (except NMI, trap, and unimplemented opcode) to be serviced by the PTS instead of the interrupt controller.

The PTS supports four special microcoded routines that enable it to complete specific tasks in much less time than an equivalent interrupt service routine can. It can transfer bytes or words, either individually or in blocks, between any memory locations in page 00H and can generate pulse-width modulated (PWM) signals. PTS interrupts have a higher priority than standard interrupts and may temporarily suspend interrupt service routines.

A block of data called the PTS control block (PTSCB) contains the specific details for each PTS routine (see "Initializing the PTS Control Blocks" on page 6-17). When a PTS interrupt occurs, the priority encoder selects the appropriate vector and fetches the PTS control block (PTSCB).

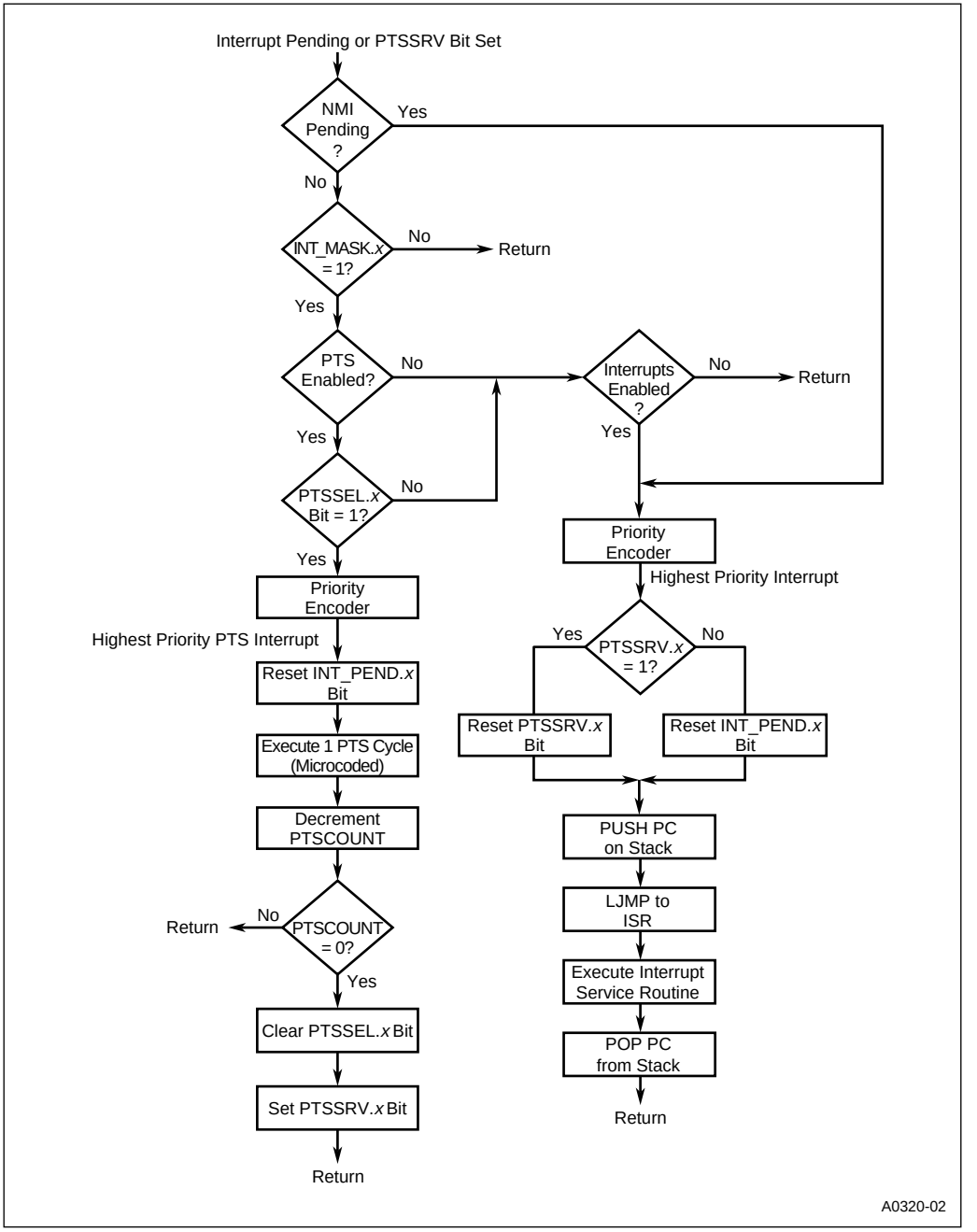


Figure 6-1. Flow Diagram for PTS and Standard Interrupts

Figure 6-1 illustrates the interrupt processing flow. In this flow diagram, “INT_MASK” represents both the INT_MASK and INT_MASK1 registers, and “INT_PEND” represents both the INT_PEND and INT_PEND1 registers.

6.2 INTERRUPT SIGNALS AND REGISTERS

Table 6-1 describes the external interrupt signals and Table 6-2 describes the control and status registers for both the interrupt controller and PTS.

Table 6-1. Interrupt Signals

Port Pin	Interrupt Signal	Type	Description
P2.2 P2.4 P3.6 P3.7	EXTINT0 EXTINT1 EXTINT2 EXTINT3	I	External Interrupts In normal operating mode, a rising edge on EXTINT*x* sets the EXTINT*x* interrupt pending bit. EXTINT*x* is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In standby and powerdown modes, asserting the EXTINT*x* signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled, but the pin must be configured as a special-function input (see “Bidirectional Port Pin Configurations” on page 7-7). If the EXTINT*x* interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation.
—	NMI	I	Nonmaskable Interrupt In normal operating mode, a rising edge on NMI generates a nonmaskable interrupt. NMI has the highest priority of all prioritized interrupts. Assert NMI for greater than one state time to guarantee that it is recognized.

Table 6-2. Interrupt and PTS Control and Status Registers

Mnemonic	Address	Description
EPA_MASK	1FA0H, 1FA1H	EPA Interrupt Mask Register This register enables/disables the four capture overrun interrupts (OVR0-3).
EPA_PEND	1FA2H, 1FA3H	EPA Interrupt Pending Register The bits in this register are set by hardware to indicate that a capture overrun has occurred.
INT_MASK INT_MASK1	0008H 0013H	Interrupt Mask Registers These registers enable/disable each maskable interrupt (that is, each interrupt except unimplemented opcode, software trap, and NMI).

Table 6-2. Interrupt and PTS Control and Status Registers (Continued)

Mnemonic	Address	Description
INT_PEND INT_PEND1	0009H 0012H	Interrupt Pending Registers The bits in this register are set by hardware to indicate that an interrupt is pending.
PSW	No direct access	Processor Status Word This register contains one bit that globally enables or disables servicing of all maskable interrupts and another that enables or disables the PTS. These bits are set or cleared by executing the enable interrupts (EI), disable interrupts (DI), enable PTS (EPTS), and disable PTS (DPTS) instructions.
PTSSEL	0004H, 0005H	PTS Select Register This register selects either a PTS routine or a standard interrupt service routine for each of the maskable interrupt requests.
PTSSRV	0006H, 0007H	PTS Service Register The bits in this register are set by hardware to request an end-of-PTS interrupt.

6.3 INTERRUPT SOURCES AND PRIORITIES

Table 6-3 lists the interrupts sources, their default priorities (30 is highest and 0 is lowest), and their vector addresses. The unimplemented opcode and software trap interrupts are not prioritized; they go directly to the interrupt controller for servicing. The priority encoder determines the priority of all other pending interrupt requests. NMI has the highest priority of all prioritized interrupts, PTS interrupts have the next highest priority, and standard interrupts have the lowest. The priority encoder selects the highest priority pending request and the interrupt controller selects the corresponding vector location in special-purpose memory. This vector contains the starting (base) address of the corresponding PTS control block (PTSCB) or interrupt service routine. PTSCBs must be located on a quad-word boundary in the internal register file. Interrupt service routines must begin execution in page FFH, but can jump anywhere after the initial vector is taken.

6.3.1 Special Interrupts

This microcontroller has three special interrupt sources that are always enabled: unimplemented opcode, software trap, and NMI. These interrupts are not affected by the EI (enable interrupts) and DI (disable interrupts) instructions, and they cannot be masked. All of these interrupts are serviced by the interrupt controller; they cannot be assigned to the PTS. Of these three, only NMI goes through the transition detector and priority encoder. The other two special interrupts go directly to the interrupt controller for servicing. Be aware that these interrupts are often assigned to special functions in development tools.

Table 6-3. Interrupt Sources, Vectors, and Priorities

Interrupt Source	Mnemonic	Interrupt Controller Service			PTS Service		
		Name	Vector	Priority	Name	Vector	Priority
Nonmaskable Interrupt	NMI	INT15	FF203EH	30	—	—	—
EXTINT3 Pin	EXTINT3	INT14	FF203CH	14	PTS14	FF205CH	29
EXTINT2 Pin	EXTINT2	INT13	FF203AH	13	PTS13	FF205AH	28
EPA capture overrun in channel 2 or 3	OVR2_3 †	INT12	FF2038H	12	PTS12	FF2058H	27
EPA capture overrun in channel 0 or 1	OVR0_1 †	INT11	FF2036H	11	PTS11	FF2056H	26
EPA Capture/Compare 3	EPA3	INT10	FF2034H	10	PTS10	FF2054H	25
EPA Capture/Compare 2	EPA2	INT09	FF2032H	09	PTS09	FF2052H	24
EPA Capture/Compare 1	EPA1	INT08	FF2030H	08	PTS08	FF2050H	23
Unimplemented Opcode	—	—	FF2012H	—	—	—	—
Software TRAP Instruction	—	—	0FF2010H	—	—	—	—
EPA Capture/Compare 0	EPA0	INT07	FF200EH	07	PTS07	FF204EH	22
SIO Receive	RI	INT06	FF200CH	06	PTS06	FF204CH	21
SIO Transmit	TI	INT05	FF200AH	05	PTS05	FF204AH	20
EXTINT1 Pin	EXTINT1	INT04	FF2008H	04	PTS04	FF2048H	19
EXTINT0 Pin	EXTINT0	INT03	FF2006H	03	PTS03	FF2046H	18
Reserved	Reserved	INT02	FF2004H	02	PTS02	FF2044H	17
Timer 2 Overflow	OVRTM2	INT01	FF2002H	01	PTS01	FF2042H	16
Timer 1 Overflow	OVRTM1	INT00	FF2000H	00	PTS00	FF2040H	15

† PTS service is not recommended because the PTS cannot determine the source of shared interrupts.

6.3.1.1 Unimplemented Opcode

If the CPU attempts to execute an unimplemented opcode, an indirect vector through location FF2012H occurs. This prevents random software execution during hardware and software failures. The interrupt vector should contain the starting address of an error routine that will not further corrupt an already erroneous situation. The unimplemented opcode interrupt prevents other interrupt requests from being acknowledged until after the next instruction is executed.

6.3.1.2 Software Trap

The TRAP instruction (opcode F7H) causes an interrupt call that is vectored through location FF2010H. The TRAP instruction provides a single-instruction interrupt that is useful when debugging software or generating software interrupts. The TRAP instruction prevents other interrupt requests from being acknowledged until after the next instruction is executed.

6.3.1.3 NMI

The external NMI pin generates a nonmaskable interrupt for implementation of critical interrupt routines. NMI has the highest priority of all the prioritized interrupts. It is passed directly from the transition detector to the priority encoder, and it vectors indirectly through location FF203EH. The NMI pin is sampled during phase 2 (CLKOUT high) and is latched internally. Because interrupts are edge-triggered, only one interrupt is generated, even if the pin is held high. If your system does not use the NMI interrupt, connect the NMI pin to V_{SS} to prevent spurious interrupts.

6.3.2 External Interrupt Pins

The external interrupt pins are multiplexed with port pins as follows: EXTINT0/P2.2, EXTINT1/P2.4, EXTINT2/P3.6, and EXTINT3/P3.7. Writing to a bit in the P*x*_MODE register also sets the corresponding external interrupt bit in the interrupt pending register. To prevent false interrupts, first configure the port pins and then clear the interrupt pending registers before globally enabling interrupts. See "Design Considerations for External Interrupt Inputs" on page 7-11.

The interrupt detection logic can generate an interrupt if a momentary negative glitch occurs while the input pin is held high. For this reason, interrupt inputs should normally be held low when they are inactive.

6.3.3 Multiplexed Interrupt Sources

The overrun errors for the four capture/compare modules are multiplexed into two interrupt pairs: OVR0_1 (channels 0 and 1) and OVR2_3 (channels 2 and 3). Generally, PTS interrupt service is not useful for multiplexed interrupts because the PTS cannot readily determine the interrupt source. Your interrupt service routine should read the EPA_PEND register to determine the source of the interrupt and to ensure that no additional interrupts are pending before executing the return instruction. Chapter 10, "Event Processor Array (EPA)," discusses the EPA interrupts in detail.

6.3.4 End-of-PTS Interrupts

When the PTSCOUNT register decrements to zero at the end of a single transfer or block transfer routine, hardware clears the corresponding bit in the PTSSEL register, which disables PTS service for that interrupt. It also sets the corresponding PTSSRV bit, requesting an end-of-PTS interrupt. An end-of-PTS interrupt has the same priority as a corresponding standard interrupt. The interrupt controller processes it with an interrupt service routine that is stored in the memory location pointed to by the standard interrupt vector. For example, the PTS services the SIO transmit inter-

rupt if PTSSEL.5 is set. The interrupt vectors through FF204AH, but the corresponding end-of-PTS interrupt vectors through FF200AH, the standard SIO transmit interrupt vector. When the end-of-PTS interrupt vectors to the interrupt service routine, hardware clears the PTSSRV bit. The end-of-PTS interrupt service routine should reinitialize the PTSCB, if required, and set the appropriate PTSSEL bit to re-enable PTS interrupt service.

6.4 INTERRUPT LATENCY

Interrupt latency is the total delay between the time that the interrupt request is generated (not acknowledged) and the time that the device begins executing either the standard interrupt service routine or the PTS interrupt service routine. A delay occurs between the time that the interrupt request is detected and the time that it is acknowledged. An interrupt request is acknowledged when the current instruction finishes executing. If the interrupt request occurs during one of the last four state times of the instruction, it may not be acknowledged until after the next instruction finishes. This additional delay occurs because instructions are prefetched and prepared a few state times before they are executed. Thus, the maximum delay between interrupt request and acknowledgment is four state times plus the execution time of the next instruction.

When a standard interrupt request is acknowledged, the hardware clears the interrupt pending bit and forces a call to the address contained in the corresponding interrupt vector. When a PTS interrupt request is acknowledged, the hardware immediately vectors to the PTSCB and begins executing the PTS routine.

6.4.1 Situations that Increase Interrupt Latency

If an interrupt request occurs while any of the following instructions are executing, the interrupt will not be acknowledged until after the **next** instruction is executed:

- the signed prefix opcode (FE) for the two-byte, signed multiply and divide instructions
- any of these eight *protected instructions*: DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, PUSHF (see Appendix A for descriptions of these instructions)
- any of the read-modify-write instructions: AND, ANDB, OR, ORB, XOR, XORB

Both the unimplemented opcode interrupt and the software trap interrupt prevent other interrupt requests from being acknowledged until after the next instruction is executed.

Each PTS cycle within a PTS routine cannot be interrupted. A PTS cycle is the entire PTS response to a single interrupt request. In block transfer mode, a PTS cycle consists of the transfer of an entire block of bytes or words. This means a worst-case latency of 500 states if you assume a block transfer of 32 words from one external memory location to another. See Table 6-4 on page 6-10 for PTS cycle execution times.

6.4.2 Calculating Latency

The maximum latency occurs when the interrupt request occurs too late for acknowledgment following the current instruction. The following worst-case calculation assumes that the current instruction is not a protected instruction. To calculate latency, add the following terms:

- Time for the current instruction to finish execution (4 state times).
  - — If this is a protected instruction, the instruction that follows it must also execute before the interrupt can be acknowledged. Add the execution time of the instruction that follows a protected instruction.
- Time for the next instruction to execute. (The longest instruction, NORML, takes 39 state times. However, the BMOV instruction could actually take longer if it is transferring a large block of data. If your code contains routines that transfer large blocks of data, you may get a more accurate worst-case value if you use the BMOV instruction in your calculation instead of NORML. See Appendix A for instruction execution times.)
- For standard interrupts only, the response time to get the vector and force the call
  - — in 64-Kbyte mode, 11 state times for an internal stack or 13 for an external stack (assuming a zero-wait-state bus)
  - — in 1-Mbyte mode, 15 state times for an internal stack or 18 for an external stack (assuming a zero-wait-state bus)

6.4.2.1 Standard Interrupt Latency

In 64-Kbyte mode, the worst-case delay for a standard interrupt is 56 state times (4 + 39 + 11 + 2) if the stack is in external memory (Figure 6-2). In 1-Mbyte mode, the worst-case delay increases to 61 state times (4 + 39 + 15 + 3) (Figure 6-2). This delay time does not include the time needed to execute the first instruction in the interrupt service routine or to execute the instruction following a protected instruction.

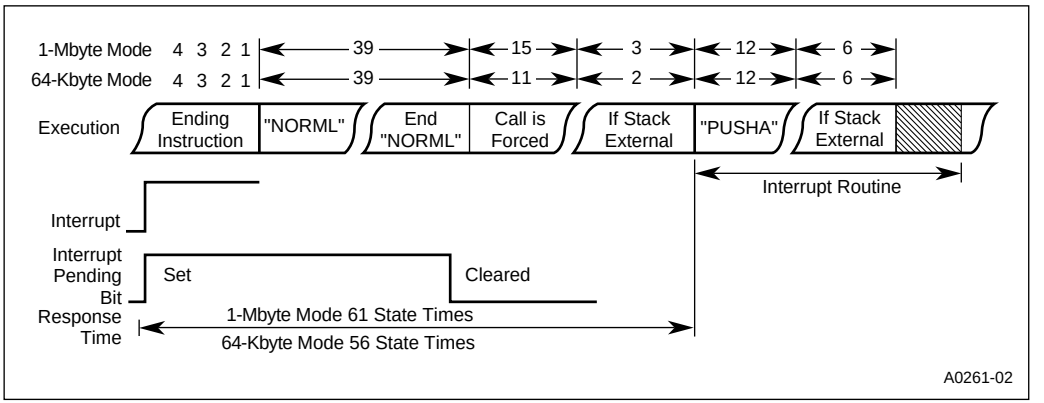


Figure 6-2. Standard Interrupt Response Time

6.4.2.2 PTS Interrupt Latency

In both 64-Kbyte and 1-Mbyte modes, the maximum delay for a PTS interrupt is 43 state times (4 + 39) as shown in Figure 6-3. This delay time does not include the added delay if a protected instruction is being executed or if a PTS request is already in progress. See Table 6-4 for execution times for PTS cycles.

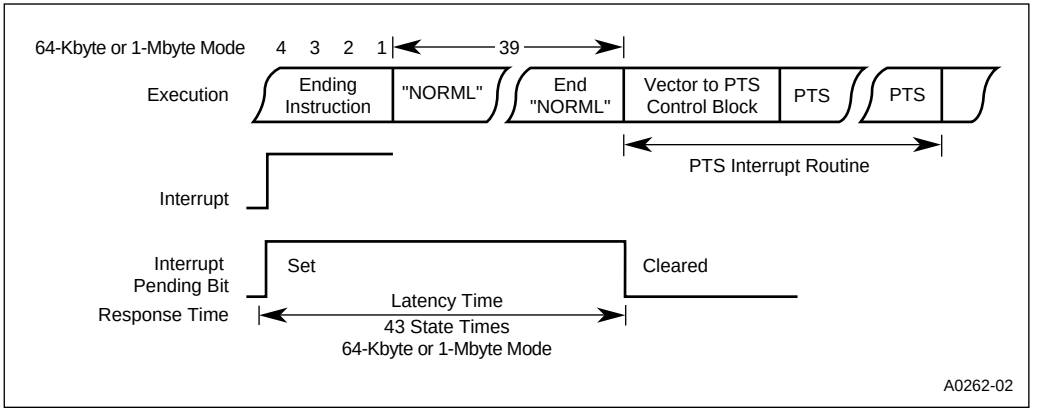


Figure 6-3. PTS Interrupt Response Time

Table 6-4. Execution Times for PTS Cycles

PTS Mode	Execution Time (in State Times)
Single transfer mode register/register† memory/register† memory/memory†	 18 per byte or word transfer + 1 21 per byte or word transfer + 1 24 per byte or word transfer + 1
Block transfer mode register/register† memory/register† memory/memory†	 13 + 7 per byte or word transfer (1 minimum) 16 + 7 per byte or word transfer (1 minimum) 19 + 7 per byte or word transfer (1 minimum)
PWM remap mode	15
PWM toggle mode	15

† *Register* indicates an access to the register file or peripheral SFR. *Memory* indicates an access to a memory-mapped register, I/O, or memory. See Table 5-1 on page 5-4 for address information.

6.5 PROGRAMMING THE INTERRUPTS

The PTS select register (PTSSEL) selects either PTS service or a standard software interrupt service routine for each of the maskable interrupt requests (see Figure 6-4). The interrupt mask registers, INT_MASK and INT_MASK1, enable or disable (mask) individual interrupts (see Figures 6-5 and 6-6). With the exception of the nonmaskable interrupt (NMI) bit (INT_MASK1.7), setting a bit enables the corresponding interrupt source and clearing a bit disables the source.

To disable any interrupt, clear its mask bit. To enable an interrupt for standard interrupt service, set its mask bit and clear its PTS select bit. To enable an interrupt for PTS service, set both the mask bit and the PTS select bit.

When you assign an interrupt to the PTS, you must set up a PTS control block (PTSCB) for each interrupt source (see "Initializing the PTS Control Blocks" on page 6-17) and use the EPTS instruction to globally enable the PTS. When you assign an interrupt to a standard software service routine, use the EI (enable interrupts) instruction to globally enable interrupt servicing.

NOTE

The DI (disable interrupts) instruction does not disable PTS service. However, it does disable service for the end-of-PTS interrupt request. If an interrupt request occurs while interrupts are disabled, the corresponding pending bit is set in the INT_PEND or INT_PEND1 register.

PTSSEL Address: 0004H
Reset State: 0000H

The PTS select (PTSSEL) register selects either a PTS microcode routine or a standard interrupt service routine for each interrupt request. Setting a bit selects a PTS microcode routine; clearing a bit selects a standard interrupt service routine. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit. The PTSSEL bit must be set manually to re-enable the PTS channel.

15							8
—	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1
7							**0**
EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM2	OVRTM1

Bit Number	Function
15, 2	Reserved; for compatibility with future devices, write zero to this bit.
14:3 1:0	Setting a bit causes the corresponding interrupt to be handled by a PTS microcode routine. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
EXTINT3	EXTINT3 pin	FF205CH
EXTINT2	EXTINT2 pin	FF205AH
OVR2_3†	EPA Capture Channel 2 or 3 Overrun	FF2058H
OVR0_1†	EPA Capture Channel 0 or 1 Overrun	FF2056H
EPA3	EPA Capture/Compare Channel 3	FF2054H
EPA2	EPA Capture/Compare Channel 2	FF2052H
EPA1	EPA Capture/Compare Channel 1	FF2050H
EPA0	EPA Capture/Compare Channel 0	FF204EH
RI	SIO Receive	FF204CH
TI	SIO Transmit	FF204AH
EXTINT1	EXTINT1 pin	FF2048H
EXTINT0	EXTINT0 pin	FF2046H
OVRTM2	Timer 2 Overflow/ Underflow	FF2042H
OVRTM1	Timer 1 Overflow/ Underflow	FF2040H

† PTS service is not recommended because the PTS cannot determine the source of shared interrupts.

Figure 6-4. PTS Select (PTSSEL) Register

6.5.1 Programming Considerations for Multiplexed Interrupts

An overrun on the EPA capture compare channels can generate the multiplexed capture overrun interrupts (OVR0_1 and OVR2_3). Write to the EPA_MASK (Figure 10-11 on page 10-22) register to enable or disable the multiplexed interrupt sources and the INT_MASK1 register to enable or disable the OVR0_1 and OVR2_3 interrupts.

PTS service is not recommended for multiplexed interrupts because it cannot determine the interrupt source.

INT_MASK Address: 0008H
Reset State: 00H

The interrupt mask (INT_MASK) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK is the low byte of the processor status word (PSW); therefore, PUSHF or PUSHA saves this register on the stack and POPF or POPA restores it.

7							0
EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM2	OVRTM1

Bit Number	Function
7:3 1:0	Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows:
2	Reserved; for compatibility with future devices, write zero to this bit.

Bit Mnemonic	Interrupt	Standard Vector
EPA0	EPA Capture/Compare Channel 0	FF200EH
RI	SIO Receive	FF200CH
TI	SIO Transmit	FF200AH
EXTINT1	EXTINT1 pin	FF2008H
EXTINT0	EXTINT0 pin	FF2006H
OVRTM2	Timer 2 Overflow/Underflow	FF2002H
OVRTM1	Timer 1 Overflow/Underflow	FF2000H

Figure 6-5. Interrupt Mask (INT_MASK) Register

INT_MASK1 Address: 0013H
Reset State: 00H

The interrupt mask 1 (INT_MASK1) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK1 can be read from or written to as a byte register. PUSHA saves this register on the stack and POPA restores it.

7							0
NMI	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1

Bit Number	Function
7:0	Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
NMI	Nonmaskable Interrupt	FF203EH
EXTINT3	EXTINT3 pin	FF203CH
EXTINT2	EXTINT2 pin	FF203AH
OVR2_3†	EPA Capture Channel 2 or 3 Overrun	FF2038H
OVR0_1†	EPA Capture Channel 0 or 1 Overrun	FF2036H
EPA3	EPA Capture/Compare Channel 3	FF2034H
EPA2	EPA Capture/Compare Channel 2	FF2032H
EPA1	EPA Capture/Compare Channel 1	FF2030H

† An overrun on the EPA capture/compare channels can generate the multiplexed capture overrun interrupts. The EPA_MASK and EPA_PEND registers decode these multiplexed interrupts. Write to EPA_MASK to enable the interrupt sources; read EPA_PEND to determine which source caused the interrupt.

Figure 6-6. Interrupt Mask 1 (INT_MASK1) Register

6.5.2 Modifying Interrupt Priorities

Your software can modify the default priorities of maskable interrupts by controlling the interrupt mask registers (INT_MASK and INT_MASK1). For example, you can specify which interrupts, if any, can interrupt an interrupt service routine. The following code shows one way to prevent all interrupts, except EXTINT3 (priority 14), from interrupting an SIO receive interrupt service routine (priority 06).

```
SERIAL_RI_ISR:
     PUSHA                          ; Save PSW, INT_MASK, INT_MASK1, & WSR
                                    ; (this disables all interrupts)
     LDB INT_MASK1, #01000000B      ; Enable EXTINT3 only
     EI                             ; Enable interrupt servicing

                                    ; Service the RI interrupt

     POPA                           ; Restore PSW, INT_MASK, INT_MASK1, &
                                    ; WSR registers
     RET
CSEG AT 0FF200CH                    ; fill in interrupt table

     DCW LSW SERIAL_RI_ISR          ; LSW is a compiler directive that means
                                    ; least-significant word of vector address

     END
```

Note that location FF200CH in the interrupt vector table must be loaded with the value of the label SERIAL_RI_ISR before the interrupt request occurs and that the receive interrupt must be enabled for this routine to execute.

This routine, like all interrupt service routines, is handled in the following manner:

1. After the hardware detects and prioritizes an interrupt request, it generates and executes an interrupt call. This pushes the program counter onto the stack and then loads it with the contents of the vector corresponding to the highest priority, pending, unmasked interrupt. The hardware will not allow another interrupt call until after the first instruction of the interrupt service routine is executed.
2. The PUSHA instruction saves the contents of the PSW, INT_MASK, INT_MASK1, and window selection register (WSR) onto the stack and then clears the PSW, INT_MASK, and INT_MASK1 registers. In addition to the arithmetic flags, the PSW contains the global interrupt enable bit (I) and the PTS enable bit (PSE). By clearing the PSW and the interrupt mask registers, PUSHA effectively masks all maskable interrupts, disables standard interrupt servicing, and disables the PTS. Because PUSHA is a protected instruction, it also inhibits interrupt calls until after the next instruction executes.
3. The LDB INT_MASK1 instruction enables those interrupts that you choose to allow to interrupt the service routine. In this example, only EXTINT3 can interrupt the receive interrupt service routine. By enabling or disabling interrupts, the software establishes its own interrupt servicing priorities.
4. The EI instruction re-enables interrupt processing and inhibits interrupt calls until after the next instruction executes.
5. The actual interrupt service routine executes within the priority structure established by the software.

6. At the end of the service routine, the POPA instruction restores the original contents of the PSW, INT_MASK, INT_MASK1, and WSR registers; any changes made to these registers during the interrupt service routine are overwritten. Because interrupt calls cannot occur immediately following a POPA instruction, the last instruction (RET) will execute before another interrupt call can occur.

Notice that the “preamble” and exit code for this routine does not save or restore register RAM. The interrupt service routine is assumed to allocate its own private set of registers from the lower register file. The general-purpose register RAM in the lower register file makes this quite practical. In addition, the RAM in the upper register file is available via *windowing* (see “Windowing” on page 5-13).

6.5.3 Determining the Source of an Interrupt

When the transition detector detects an interrupt, it sets the corresponding bit in the INT_PEND or INT_PEND1 register (Figures 6-7 and 6-8). This bit is set even if the individual interrupt is disabled (masked). The pending bit is cleared when the program vectors to the interrupt service routine. INT_PEND and INT_PEND1 can be read, to determine which interrupts are pending. They can also be modified (written), either to clear pending interrupts or to generate interrupts under software control. However, we recommend the use of the read-modify-write instructions, such as AND and OR, to modify these registers.

```
ANDB INT_PEND, #11111110B          ; Clears the OVRTM1 pending bit
ORB  INT_PEND, #00000001B          ; Sets the OVRTM1 pending bit
```

Other methods could result in a partial interrupt cycle. For example, an interrupt could occur during an instruction sequence that loads the contents of the interrupt pending register into a temporary register, modifies the contents of the temporary register, and then writes the contents of the temporary register back into the interrupt pending register. If the interrupt occurs during one of the last four states of the second instruction, it will not be acknowledged until after the completion of the third instruction. Because the third instruction overwrites the contents of the interrupt pending register, the jump to the interrupt vector will not occur.

An overrun on the EPA capture compare channels can generate the multiplexed capture overrun interrupts (OVR0_1 and OVR2_3). Read the EPA_PEND register to determine the source of the interrupt request (Figure 10-12 on page 10-23).

INT_PEND Address: 0009H
Reset State: 00H

When hardware detects a pending interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7							0
EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM2	OVRTM1

Bit Number	Function
7:3 1:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. The standard interrupt vector locations are as follows: **Bit Mnemonic** — **Interrupt** — **Standard Vector** EPA0 — EPA Capture/Compare Channel 0 — FF200EH RI — SIO Receive — FF200CH TI — SIO Transmit — FF200AH EXTINT1 — EXTINT1 pin — FF2008H EXTINT0 — EXTINT0 pin — FF2006H OVRTM2 — Timer 2 Overflow/Underflow — FF2002H OVRTM1 — Timer 1 Overflow/Underflow — FF2000H
2	Reserved. This bit is undefined.

Figure 6-7. Interrupt Pending (INT_PEND) Register

INT_PEND1 Address: 0012H
Reset State: 00H

When hardware detects a pending interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7							0
NMI	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1

Bit Number	Function
7:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. The standard interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	Standard Vector
NMI	Nonmaskable Interrupt	FF203EH
EXTINT3	EXTINT3 pin	FF203CH
EXTINT2	EXTINT2 pin	FF203AH
OVR2_3†	EPA Capture Channel 2 or 3 Overrun	FF2038H
OVR0_1†	EPA Capture Channel 0 or 1 Overrun	FF2036H
EPA3	EPA Capture/Compare Channel 3	FF2034H
EPA2	EPA Capture/Compare Channel 2	FF2032H
EPA1	EPA Capture/Compare Channel 1	FF2030H

† An overrun on the EPA capture/compare channels can generate the multiplexed capture overrun interrupts. The EPA_MASK and EPA_PEND registers decode these multiplexed interrupts. Write to EPA_MASK to enable the interrupt sources; read EPA_PEND to determine which source caused the interrupt.

Figure 6-8. Interrupt Pending 1 (INT_PEND1) Register

6.6 INITIALIZING THE PTS CONTROL BLOCKS

Each PTS interrupt requires a block of data, in register RAM, called the PTS control block (PTSCB). The PTSCB identifies which PTS microcode routine will be invoked and sets up the specific parameters for the routine. You must set up the PTSCB for each interrupt source **before** enabling the corresponding PTS interrupts.

The address of the first (lowest) PTSCB byte is stored in the PTS vector table in special-purpose memory (see "Special-purpose Memory" on page 5-6). Figure 6-9 shows the PTSCB for each PTS mode. Unused PTSCB bytes can be used as extra RAM.

NOTE

The PTSCB must be located in the internal register file. The location of the first byte of the PTSCB must be aligned on a quad-word boundary (an address evenly divisible by 8). Because the PTS uses nonextended addressing, it cannot operate across page boundaries. For example, PTSSRC cannot point to a location on page 05 while PTSDST points to page 00. In the 8XC196NP, all nonextended data accesses will operate from the page defined by EP_REG. For PTS routines, write 00H to EP_REG to select page 00H (see "Accessing Data" on page 5-23). The 80C196NU forces all nonextended data accesses to page 00H. You cannot use EP_REG to change pages.

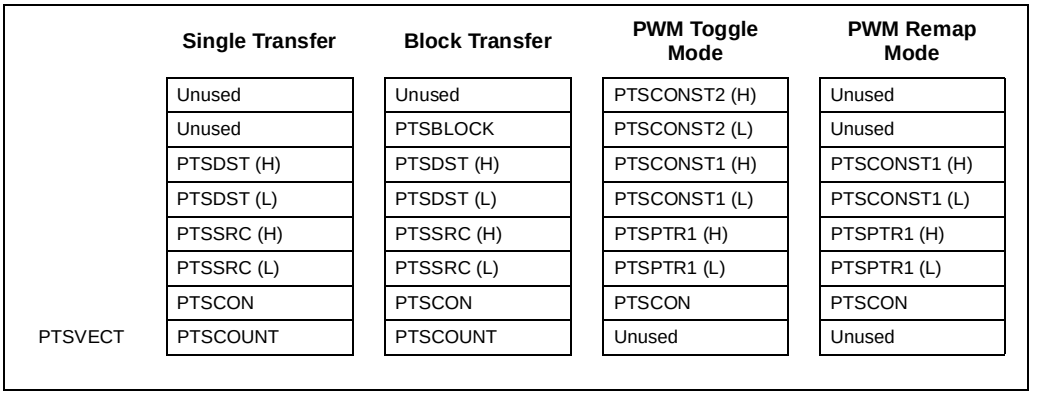


Figure 6-9. PTS Control Blocks

6.6.1 Specifying the PTS Count

For single and block transfer routines, the first location of the PTSCB contains an 8-bit value called PTSCOUNT. This value defines the number of interrupts that will be serviced by the PTS routine. The PTS decrements PTSCOUNT after each PTS cycle. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit (Figure 6-10), which requests an end-of-PTS interrupt. The end-of-PTS interrupt service routine should reinitialize the PTSCB, if required, and set the appropriate PTSSEL bit to re-enable PTS interrupt service.

PTSSRV

Address: 0006H
Reset State: 0000H

The PTS service (PTSSRV) register is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PTSSRV bit. The PTSSEL bit must be set manually to re-enable the PTS channel.

15							8
—	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1
7							**0**
EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM1	OVRTM2

Bit Number	Function
15, 2	Reserved. These bits are undefined.
14:3 1:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows.

Bit Mnemonic	Interrupt	Standard Vector
EXTINT3	EXTINT3 Pin	FF203CH
EXTINT2	EXTINT2 Pin	FF203AH
OVR2_3†	EPA Capture Channel 2 or 3 Overrun	FF2038H
OVR0_1†	EPA Capture Channel 0 or 1 Overrun	FF2036H
EPA3	EPA Capture/Compare Channel 3	FF2034H
EPA2	EPA Capture/Compare Channel 2	FF2032H
EPA1	EPA Capture/Compare Channel 1	FF2030H
EPA0	EPA Capture/Compare Channel 0	FF200EH
RI	SIO Receive	FF200CH
TI	SIO Transmit	FF200AH
EXTINT1	EXTINT1 pin	FF2008H
EXTINT0	EXTINT0 pin	FF2006H
OVRTM2	Timer 2 Overflow/Underflow	FF2002H
OVRTM1	Timer 1 Overflow/Underflow	FF2000H

† PTS service is not recommended for multiplexed interrupts. This bit is cleared when both corresponding interrupt pending bits are cleared in EPA_PEND.

Figure 6-10. PTS Service (PTSSRV) Register

6.6.2 Selecting the PTS Mode

The second byte of each PTSCB is always an 8-bit value called PTSCON. Bits 5–7 select the PTS mode (Figure 6-11). The function of bits 0–4 differ for each PTS mode. Refer to the sections that describe each mode in detail to see the function of these bits. Table 6-4 on page 6-10 lists the cycle execution times for each PTS mode.

PTSCON Address: PTSPCB + 1

The PTS control (PTSCON) register selects the PTS mode and sets up control functions for that mode.

7							0
M2	M1	M0	†	†	†	†	†

Bit Number	Bit Mnemonic	Function
7:5	M2:0	PTS Mode These bits select the PTS mode: **M2 M1 M0** 0 0 0 block transfer 0 0 1 reserved 0 1 0 PWM toggle or remap 0 1 1 reserved 1 0 0 single transfer 1 0 1 reserved 1 1 0 reserved 1 1 1 reserved

† The function of this bit depends upon which mode is selected. See the PTS control block description in each PTS mode section.

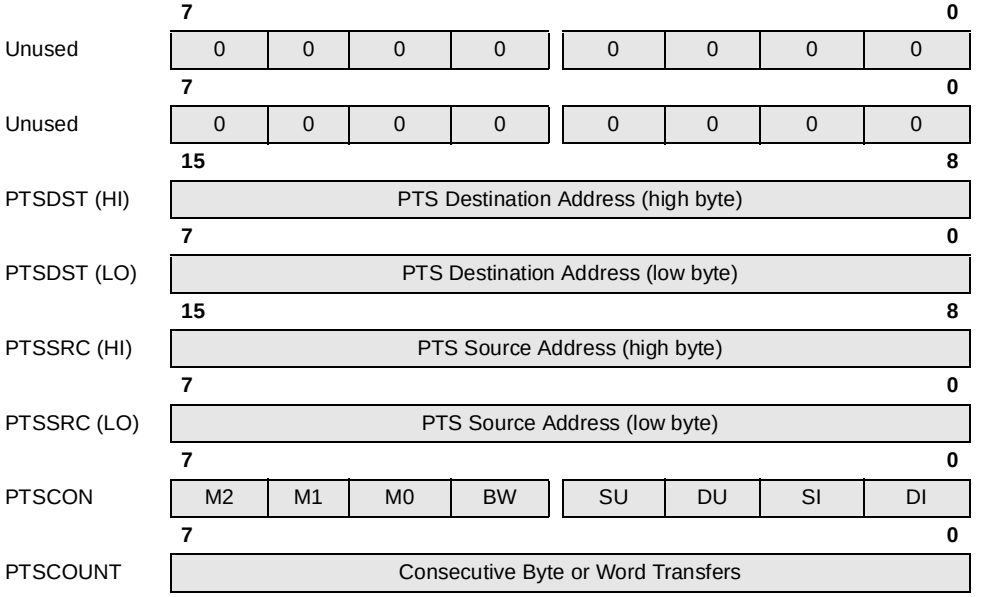
Figure 6-11. PTS Mode Selection Bits (PTSCON Bits 7:5)

6.6.3 Single Transfer Mode

In single transfer mode, an interrupt causes the PTS to transfer a single byte or word (selected by the BW bit in PTSCON) from one memory location to another. This mode is typically used with serial I/O or synchronous serial I/O interrupts. It can also be used with the EPA to move captured time values from the event-time register to internal RAM for further processing. See AP-445, *8XC196KR Peripherals: A User's Point of View*, for application examples with code. Figure 6-12 shows the PTS control block for single transfer mode.

PTS Single Transfer Mode Control Block

In single transfer mode, the PTS control block contains a source and destination address (PTSSRC and PTSDST), a control register (PTSCON), and a transfer count (PTSCOUNT).



Register	Location	Function
PTSDST	PTSCB + 4	PTS Destination Address Write the destination memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.
PTSSRC	PTSCB + 2	PTS Source Address Write the source memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.

Figure 6-12. PTS Control Block — Single Transfer Mode

PTS Single Transfer Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode **M2 M1 M0** 1 0 0 single transfer mode
		BW	Byte/Word Transfer 0 = word transfer 1 = byte transfer
		SU†	Update PTSSRC 0 = reload original PTS source address after each byte or word transfer 1 = retain current PTS source address after each byte or word transfer
		DU†	Update PTSDST 0 = reload original PTS destination address after each byte or word transfer 1 = retain current PTS destination address after each byte or word transfer
		SI†	PTSSRC Autoincrement 0 = do not increment the contents of PTSSRC after each byte or word transfer 1 = increment the contents of PTSSRC after each byte or word transfer
		DI†	PTSDST Autoincrement 0 = do not increment the contents of PTSDST after each byte or word transfer 1 = increment the contents of PTSDST after each byte or word transfer
PTSCOUNT	PTSCB + 0	Consecutive Word or Byte Transfers Defines the number of words or bytes that will be transferred during the single transfer routine. Each word or byte transfer is one PTS cycle. Maximum value is 255.	

† The DU/DI bits and SU/SI bits are paired in single transfer mode. Each pair must be set or cleared together. However, the two pairs, DU/DI and SU/SI, need not be equal.

Figure 6-12. PTS Control Block — Single Transfer Mode (Continued)

The PTSCB in Table 6-5 defines nine PTS cycles. Each cycle moves a single word from location 20H to an external memory location. The PTS transfers the first word to location 6000H. Then it increments and updates the destination address and decrements the PTSCOUNT register; it does not increment the source address. When the second cycle begins, the PTS moves a second word from location 20H to location 6002H. When PTSCOUNT equals zero, the PTS will have filled locations 6000–600FH, and an end-of-PTS interrupt is generated.

Table 6-5. Single Transfer Mode PTSCB

Unused
Unused
PTSDST (HI) = 60H
PTSDST (LO) = 00H
PTSSRC (HI) = 00H
PTSSRC (LO) = 20H
PTSCON = 85H (Mode = 100, BW = 0, SI/SU = 0, DI/DU = 1)
PTSCOUNT = 09H

6.6.4 Block Transfer Mode

In block transfer mode, an interrupt causes the PTS to move a block of bytes or words from one memory location to another. See AP-445, *8XC196KR Peripherals: A User's Point of View*, for application examples with code. Figure 6-13 shows the PTS control block for block transfer modes.

In this mode, each PTS cycle consists of the transfer of an entire block of bytes or words. Because a PTS cycle cannot be interrupted, the block transfer mode can create long interrupt latency. The worst-case latency could be as high as 500 states, if you assume a block transfer of 32 words from one external memory location to another, using an 8-bit bus with no wait states. See Table 6-4 on page 6-10 for execution times of PTS cycles.

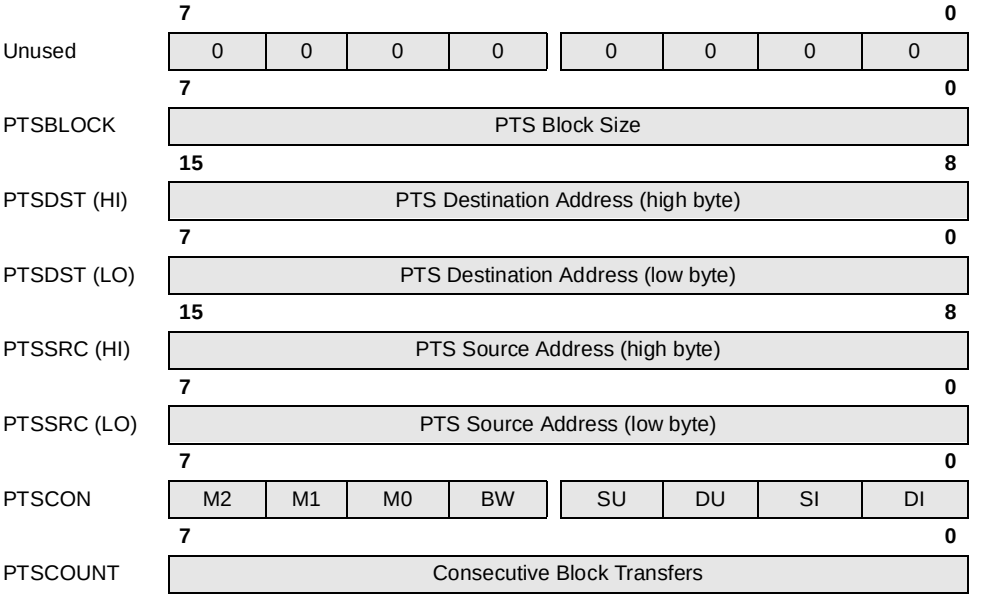
The PTSCB in Table 6-6 sets up three PTS cycles that will transfer five bytes from memory locations 20–24H to 6000–6004H (cycle 1), 6005–6009H (cycle 2), and 600A–600EH (cycle 3). The source and destination are incremented after each byte transfer, but the original source address is reloaded into PTSSRC at the end of each block-transfer cycle. In this routine, the PTS always gets the first byte from location 20H.

Table 6-6. Block Transfer Mode PTSCB

Unused
PTSBLOCK = 05H
PTSDST (HI) = 60H
PTSDST (LO) = 00H
PTSSRC (HI) = 00H
PTSSRC (LO) = 20H
PTSCON = 17H (Mode = 000; DI, SI, DU, BW = 1; SU = 0)
PTSCOUNT = 03H

PTS Block Transfer Mode Control Block

In block transfer mode, the PTS control block contains a block size (PTSBLOCK), a source and destination address (PTSSRC and PTSDST), a control register (PTSCON), and a transfer count (PTSCOUNT).



Register	Location	Function
PTSBLOCK	PTSCB + 6	PTS Block Size Specifies the number of bytes or words in each block. Valid values are 1–32, inclusive.
PTSDST	PTSCB + 4	PTS Destination Address Write the destination memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.
PTSSRC	PTSCB + 2	PTS Source Address Write the source memory location to this register. A valid address is any unreserved memory location within page 00H; however, it must point to an even address if word transfers are selected.

Figure 6-13. PTS Control Block — Block Transfer Mode

PTS Block Transfer Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits select the PTS mode: **M2 M1 M0** 0 0 0 block transfer mode
		BW	Byte/Word Transfer 0 = word transfer 1 = byte transfer
		SU	Update PTSSRC 0 = reload original PTS source address after each block transfer is complete 1 = retain current PTS source address after each block transfer is complete
		DU	Update PTSDST 0 = reload original PTS destination address after each block transfer is complete 1 = retain current PTS destination address after each block transfer is complete
		SI	PTSSRC Autoincrement 0 = do not increment the contents of PTSSRC after each byte or word transfer 1 = increment the contents of PTSSRC after each byte or word transfer
		DI	PTSDST Autoincrement 0 = do not increment the contents of PTSDST after each byte or word transfer 1 = increment the contents of PTSDST after each byte or word transfer
PTSCOUNT	PTSCB + 0	Consecutive Block Transfers Defines the number of blocks that will be transferred during the block transfer routine. Each block transfer is one PTS cycle. Maximum number is 255.	

Figure 6-13. PTS Control Block — Block Transfer Mode (Continued)

6.6.5 PWM Modes

The PWM toggle and PWM remap modes are designed for use with the event processor array (EPA) to generate pulse-width modulated (PWM) output signals. These modes can also be used with an interrupt signal from any other source. The PWM toggle mode uses a single EPA channel to generate a PWM signal. The PWM remap mode uses two EPA channels, but it can generate signals with duty cycles closer to 0% or 100% than are possible with the PWM toggle mode. Table 6-7 compares the two PWM modes. For code examples, see AP-445, *8XC196KR Peripherals: A User's Point of View,* and "EPA PWM Output Program" on page 10-26.

Table 6-7. Comparison of PWM Modes

PWM Toggle Mode	PWM Remap Mode
Uses a single EPA channel.	Uses two EPA channels.
Reads the location specified by PTSPTR1 (usually EPA*x*_TIME).	Reads the location specified by PTSPTR1 (usually EPA*x*_TIME).
Adds one of two values to the location specified by PTSPTR1. If TBIT is clear, it adds the value in PTSCONST1. If TBIT is set, it adds the value in PTSCONST2.	Adds the value in PTSCONST1 to the location specified by PTSPTR1.
Stores the sum back into the location specified by PTSPTR1.	Stores the sum back into the location specified by PTSPTR1.
Toggles TBIT.	Toggles the unused TBIT.

Figure 6-14 illustrates a generic PWM waveform. The length of an entire PWM output pulse is T2. The time the output is "on" is T1; the time the output is "off" is T2 – T1. The formulas for frequency and duty cycle are shown below. In most applications, the frequency is held constant and the duty cycle is varied to change the average value of the waveform.

$$\text{Frequency, in Hertz} = \frac{1}{T2}$$

$$\text{Duty Cycle} = \frac{T1}{T2} \times 100\%$$

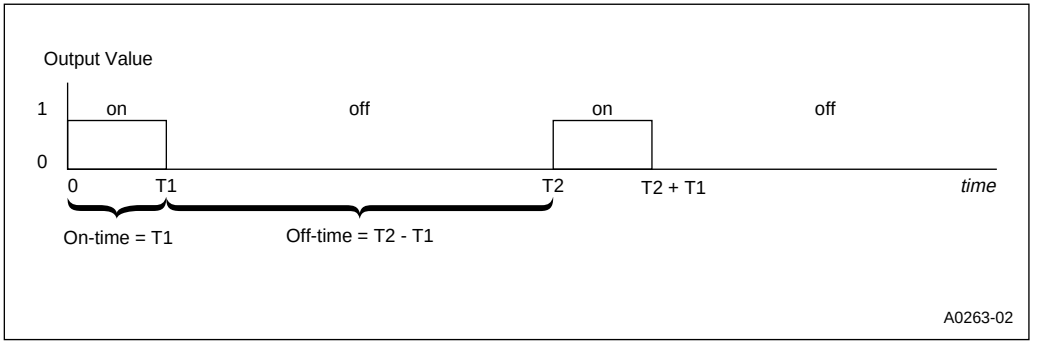


Figure 6-14. A Generic PWM Waveform

The PWM modes do not use a PTSCOUNT register to specify the number of consecutive PTS cycles. To stop producing the PWM output, first clear the PTSSEL.*x* bit to disable PTS service for the interrupt and then use the interrupt service routine to reconfigure the EPA channel.

6.6.5.1 PWM Toggle Mode Example

Figure 6-15 shows the PTS control block for PWM toggle mode. To generate a PWM waveform using PWM toggle mode and EPA0, complete the following procedure. This example uses the values stored in CSTORE1 and CSTORE2 to control the frequency and duty cycle of a PWM.

1. Disable the interrupts and the PTS. The DI instruction disables all standard interrupts; the DPTS instruction disables the PTS.
2. Store the on-time value (T1) in CSTORE1.
3. Store the off-time value (T2 – T1) in CSTORE2.
4. Set up the PTSCB as shown in Table 6-8.
   — Load PTSCON with 43H (selects PWM toggle mode, initial TBIT value = 1).
   — Set up PTSPTR1 to point to EPA0_TIME (the EPA0 event-time register).
   — Load PTSCONST1 with the on-time value (T1) from CSTORE1.
   — Load PTSCONST2 with the off-time value (T2 – T1) from CSTORE2.

Table 6-8. PWM Toggle Mode PTSCB

PTSCONST2 (HI) = T2 – T1 (HI)
PTSCONST2 (LO) = T2 – T1 (LO)
PTSCONST1 (HI) = T1 (HI)
PTSCONST1 (LO) = T1 (LO)
PTSPTR1 (HI) = 1FH
PTSPTR1 (LO) = 82H
PTSCON = 43H (Mode = 010, TMOD = 1, TBIT = 1)
Unused

5. Configure P1.0 to serve as the EPA0 output.
   — Clear P1_DIR.0 (selects output).
   — Set P1_MODE.0 (selects the EPA0 special-function signal).
   — Set P1_REG.0 (initializes the output to "1").
6. Set up EPA0.
   — Load EPA0_CON with 0078H (timer 1, compare, toggle output pin, re-enable).
   — Load EPA0_TIME with the value in PTSCONST1 (selects T1 as first event time).
   — Load T1CONTROL with C2H (enables timer 1, selects up counting at f/4, and enables the divide-by-four prescaler).
7. Enable the EPA0 interrupt and select PTS service for it.
   — Set INT_MASK.7.
   — Set PTSSEL.7.
8. Enable the interrupts and the PTS. The EI instruction enables interrupts; the EPTS instruction enables the PTS.

PTS PWM Toggle Mode Control Block

In PWM toggle mode, the PTS uses a single EPA channel to generate a pulse-width modulated (PWM) output signal. The control block contains registers that contain the PWM on-time (PTSCONST1), the PWM off-time (PTSCONST2), the address pointer (PTSPTR1), and a control register (PTSCON).

	7							0
PTSCONST2 (H)	PWM Off-time (high byte)							
	7							**0**
PTSCONST2 (L)	PWM Off-time (low byte)							
	15							**8**
PTSCONST1 (H)	PWM On-time (high byte)							
	7							**0**
PTSCONST1 (L)	PWM On-time (low byte)							
	15							**8**
PTSPTR1 (H)	Pointer 1 Value (high byte)							
	7							**0**
PTSPTR1 (L)	Pointer 1 Value (low byte)							
	7							**0**
PTSCON	M2	M1	M0	—	—	—	TMOD	TBIT
	7							**0**
Unused	0	0	0	0	0	0	0	0

Register	Location	Function
PTSCONST2	PTSCB + 6	PWM Off-time Write the desired PWM off-time to these bits.
PTSCONST1	PTSCB + 4	PWM On-time Write the desired PWM on-time to these bits.
PTSPTR1	PTSCB + 2	Pointer 1 Value These bits point to a memory location, usually EPA*x*_TIME. PTSPTR1 can point to any unreserved memory location within page 00H.

Figure 6-15. PTS Control Block — PWM Toggle Mode

PTS PWM Toggle Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits specify the PTS mode: **M2 M1 M0** 0 1 0 PWM
		TMOD	Toggle Mode Select 1 = PWM toggle mode
		TBIT	Toggle Bit Initial Value Defines the initial value of TBIT. 0 = selects initial value as zero 1 = selects initial value as one The TBIT value determines whether PTSCONST1 or PTSCONST2 is added to the PTSPTR1 value: 0 = PTSCONST1 is added to PTSPTR1 1 = PTSCONST2 is added to PTSPTR1 Reading this bit returns the current value of TBIT, which is toggled by hardware at the end of each PWM toggle cycle.

Figure 6-15. PTS Control Block — PWM Toggle Mode (Continued)

Figure 6-16 is a flow diagram of the EPA and PTS operations for this example. Operation begins when the timer is enabled (at *time* = 0 in Figure 6-14 on page 6-27) by the write to T1CONTROL. The first timer match occurs at *time* = T1. The EPA toggles the output pin to zero and generates an interrupt to initiate the first PTS cycle.

PWM Toggle Cycle 1. Because TBIT is initialized to one, the PTS adds the off-time value (T2 – T1) to EPA0_TIME and toggles TBIT to zero.

The second timer match occurs at *time* = T2 (the end of one complete PWM pulse). The EPA toggles the output to one and generates an interrupt to initiate the second PTS cycle.

PWM Toggle Cycle 2. Because TBIT is zero, the PTS adds the on-time value (T1) to EPA0_TIME and toggles the TBIT to one.

The next timer match occurs at *time* = T2 + T1. The EPA toggles the output to zero and initiates the third PTS cycle. The PTS actions are the same as in cycle 1, and generation of the PWM output continues with PTS cycle 1 and cycle 2 alternating.

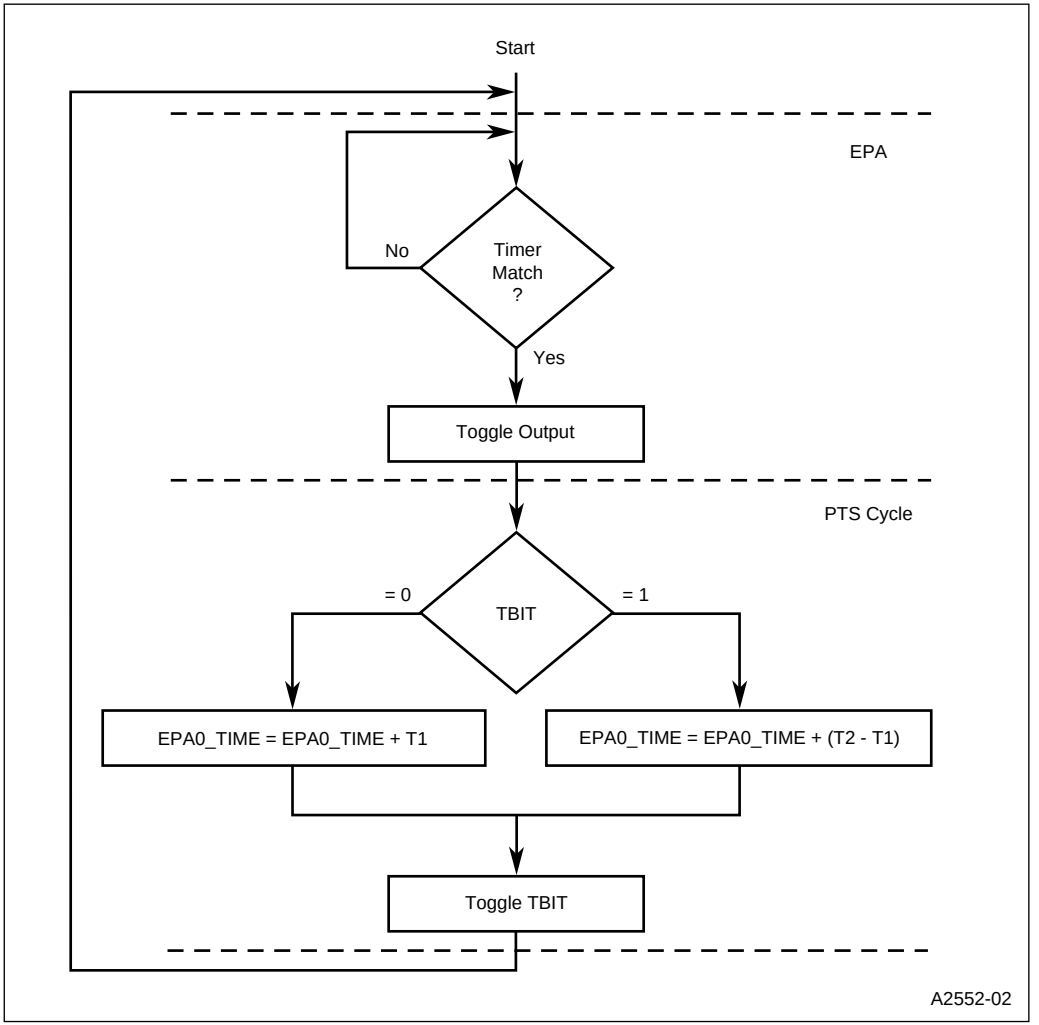


Figure 6-16. EPA and PTS Operations for the PWM Toggle Mode Example

You can modify the duty cycle without interrupting the PWM operation. To change the duty cycle during a PWM cycle, the PTS service routine should write new T1 and T2 – T1 values to CSTORE1 and CSTORE2 and select normal interrupt service for the next EPA0 interrupt. When the next timer match occurs, the output is toggled, and the device executes a normal interrupt service routine, which performs these operations:

1. The routine writes the new value of T1 (in CSTORE1) to PTSCONST1 and the new value of T1 – T2 (in CSTORE2) to PTSCONST2.
2. It selects PTS service for the EPA0 interrupt.

When the next timer match occurs, the PTS cycle (Figure 6-16) increments EPA0_TIME by T1 (if TBIT is zero (output = 0)) or T2 – T1 (if TBIT is one (output = 1)). (Note that although the values of the EPA0 output and TBIT are the same in this example, these two values are unrelated. To establish the initial value of the output, set or clear P1_REG.*x*.)

The PWM toggle mode has the advantage of using only one EPA channel. However, if the waveform edges are close together, the PTS may take too long and miss setting up the next edge. The PWM remap mode uses two EPA channels to eliminate this problem.

6.6.5.2 PWM Remap Mode Example

Figure 6-17 shows the PTS control block for PWM remap mode. The following example uses two EPA channels and a single timer to generate a PWM waveform in PWM remap mode. EPA0 asserts the output, and EPA1 deasserts it. For each channel, an interrupt is generated every T2 period, but the comparison times for the channels are offset by the on-time, T1 (see Figure 6-14 on page 6-27). Although TBIT is toggled at the end of every PWM remap mode cycle (see Table 6-7 on page 6-26), it plays no role in this mode. To generate a PWM waveform, follow this procedure.

1. Disable the interrupts and the PTS. The DI instruction disables all interrupts; the DPTS instruction disables the PTS.
2. Set up one PTSCB for EPA0 and one for EPA1 as shown in Table 6-9. Note that the two blocks are identical, except that PTSPTR1 points to EPA0_TIME for EPA0 and to EPA1_TIME for EPA1.
3. Configure P1.1 to serve as the EPA1 output. (Because EPA0 is not used as an output, port pin P1.0 can be used for standard I/O.)
   — Clear P1_DIR.1 (selects output).
   — Set P1_MODE.1 (selects the EPA0 special-function signal).
   — Set P1_REG.1 (initializes the output to "1").

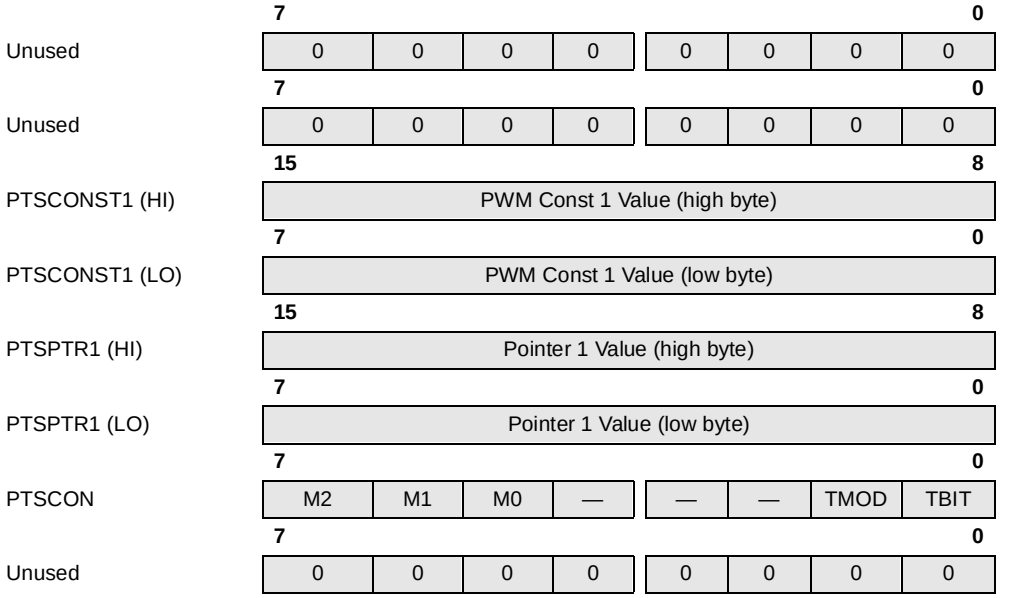
Table 6-9. PWM Remap Mode PTSCB

PTSCB0 for EPA0	PTSCB1 for EPA1
Unused	Unused
Unused	Unused
PTSCONST1 (HI) = T2 (HI)	PTSCONST1 (HI) = T2 (HI)
PTSCONST1 (LO) = T2 (LO)	PTSCONST1 (LO) = T2 (LO)
PTSPTR1 (HI) = 1FH (EPA0_TIME, HI)	PTSPTR1 (HI) = 1FH (EPA1_TIME, HI)
PTSPTR1 (LO) = 82H (EPA0_TIME, LO)	PTSPTR1 (LO) = 86H (EPA1_TIME, LO)
PTSCON = 40H (Mode = 010, TMOD = 0)	PTSCON = 40H (Mode = 010, TMOD = 0)
Unused	Unused

4. Set up EPA0 and EPA1.
   - — Load EPA0_CON with 68H (timer 1, compare mode, assert output pin, re-enable).
   - — Load EPA1_CON with 158H (timer 1, compare mode, deassert output pin, re-enable, remap enabled).
   - — Load EPA0_TIME with 0000H (selects time 0 as first event time for EPA0).
   - — Load EPA1_TIME with the value of T1 (selects time T1 as first event time for EPA1).
   - — Load timer 1 with FFFFH to ensure that the EPA0 event time (*time* = 0) is matched first.
   - — Load T1CONTROL with C2H (enables timer 1, selects up-counting at f/4, and enables the divide-by-four prescaler).
5. Enable the EPA0 and EPA1 interrupts and select PTS service for them.
   - — Set INT_MASK.7 and INT_MASK1.0.
   - — Set PTSSEL.7 and PTSSEL.8.
6. Enable the interrupts and the PTS. The EI instruction enables interrupts; the EPTS instruction enables the PTS.

PTS PWM Remap Mode Control Block

In PWM remap mode, the PTS uses two EPA channels to generate a pulse-width modulated (PWM) output signal. The control block contains registers that contain the PWM on-time (PTSCONST1), the address pointer (PTSPTR1), and a control register (PTSCON).



Register	Location	Function
PTSCONST1	PTSCB + 4	PWM Const 1 Value Write the desired PWM on-time to these bits.
PTSPTR1	PTSCB + 2	Pointer 1 Value These bits point to a memory location, usually EPA*x*_TIME. PTSPTR1 can point to any unreserved memory location within page 00H.

Figure 6-17. PTS Control Block — PWM Remap Mode

PTS PWM Remap Mode Control Block (Continued)

Register	Location	Function	
PTSCON	PTSCB + 1	PTS Control Bits	
		M2:0	PTS Mode These bits specify the PTS mode: **M2 M1 M0** 0 1 0 PWM
		TMOD	Remap Mode Select 0 = PWM remap mode
		TBIT	Toggle Bit Initial Value Defines the initial value of TBIT. 1 = selects initial value as one 0 = selects initial value as zero **NOTE:** In PWM remap mode, the TBIT value is not used; PTSCONST1 is always added to the PTSPTR1 value. However, the unused TBIT still toggles at the end of each PWM remap cycle. Reading this bit returns the current value of TBIT.

Figure 6-17. PTS Control Block — PWM Remap Mode (Continued)

Figure 6-18 shows the EPA and PTS operations for this example. The first timer match occurs at *time* = 0 for EPA0, which asserts the output and generates an interrupt.

PWM Remap Cycle 1. The PTS adds T2 to EPA0_TIME and toggles the TBIT.

The output remains asserted until the second timer match occurs at T1 for EPA1, which deasserts the output and generates an interrupt.

PWM Remap Cycle 2. The PTS adds T2 to EPA1_TIME and toggles the TBIT.

Alternating EPA0 and EPA1 interrupts continue, with EPA0 asserting the output and EPA1 deasserting it.

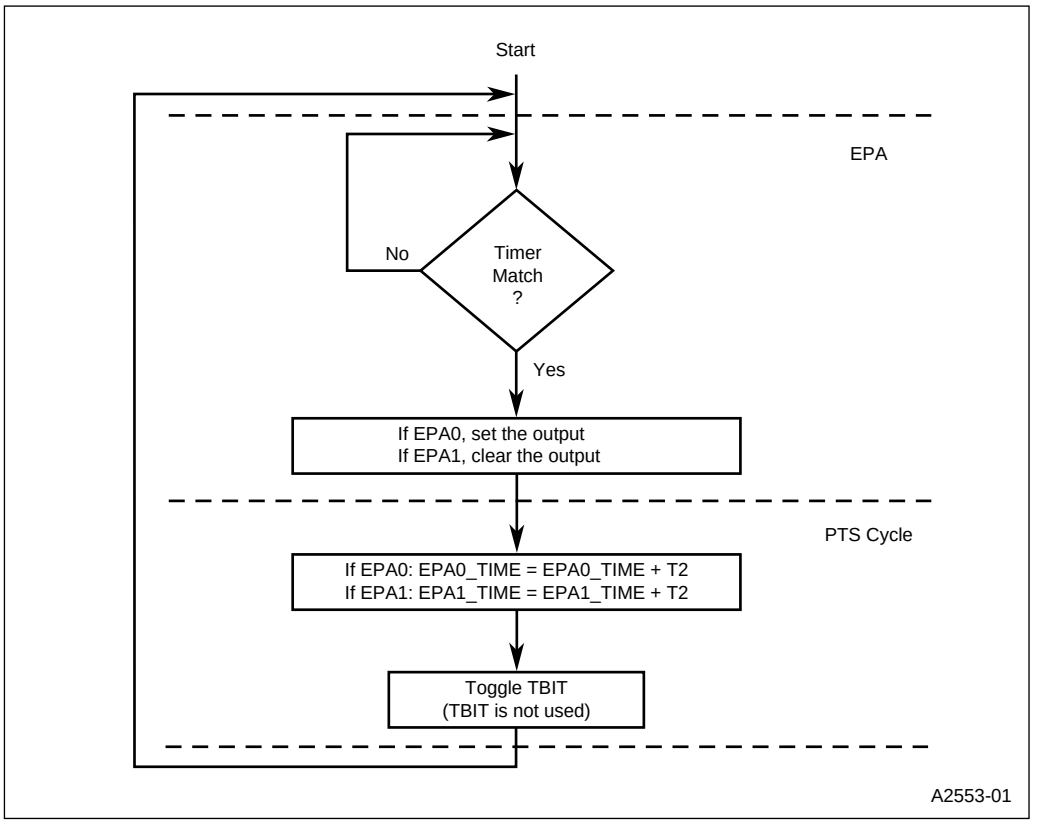


Figure 6-18. EPA and PTS Operations for the PWM Remap Mode Example

You can change the duty cycle by changing the time that the output is high and keeping the period constant. After a timer match occurs for EPA1 (when the output falls), schedule the next EPA1 match for T2 + DT, where DT is the time to be added to the on-time. Thereafter, schedule the next EPA1 match for T2. You can do this by replacing one EPA1 PTS interrupt with a normal interrupt (clear PTSSEL.8). Have the interrupt service routine add T2 + DT to EPA1_TIME and set PTSSEL.8 to re-enable PTS service for EPA1. This adjustment changes the duty cycle without affecting the period.

By using two EPA channels in the PWM remap mode, you can generate duty cycles closer to 0% and 100% than is possible with PWM toggle mode. For further information about generating PWM waveforms with the EPA, see "Operating in Compare Mode" on page 10-12.

7

I/O Ports

CHAPTER 7
I/O PORTS

I/O ports provide a mechanism to transfer information between the device and the surrounding system circuitry. They can read system status, monitor system operation, output device status, configure system options, generate control signals, provide serial communication, and so on. Their usefulness in an application is limited only by the number of I/O pins available and the imagination of the engineer.

7.1 I/O PORTS OVERVIEW

Standard I/O port registers are located in the SFR address space and they can be windowed. Memory-mapped I/O port registers are located in memory-mapped address space. Memory-mapped registers must be accessed with indirect or indexed addressing; they cannot be windowed. All ports can provide low-speed input/output pins or serve alternate functions. Table 7-1 provides an overview of the device I/O ports. The remainder of this chapter describes the ports in more detail and explains how to configure the pins. The chapters that cover the associated peripherals discuss using the pins for their special functions.

Table 7-1. Device I/O Ports

Port	Bits	Type	Direction	Associated Peripheral(s)
Port 1	8	Standard	Bidirectional	EPA and timers
Port 2	8	Standard	Bidirectional	SIO, interrupts, bus control, clock gen.
Port 3	8	Standard	Bidirectional	Chip-select unit, interrupts
Port 4	4	Standard	Bidirectional	PWM
EPORT	4	Memory mapped (NP) Standard (NU)	Bidirectional	Extended address lines

7.2 BIDIRECTIONAL PORTS 1–4

The bidirectional ports are very similar in both circuitry and configuration. All ports use Schmitt-triggered input buffers for improved noise immunity. Table 7-2 lists the bidirectional port pins with their special-function signals and associated peripherals.

Table 7-2. Bidirectional Port Pins

Port Pin	Special-function Signal(s)	Special-function Signal Type	Associated Peripheral
P1.0	EPA0	I/O	EPA
P1.1	EPA1	I/O	EPA
P1.2	EPA2	I/O	EPA
P1.3	EPA3	I/O	EPA
P1.4	T1CLK	I	Timer 1
P1.5	T1DIR	I	Timer 1
P1.6	T2CLK	I	Timer 2
P1.7	T2DIR	I	Timer 2
P2.0	TXD	O	SIO
P2.1	RXD	I/O	SIO
P2.2	EXTINT0	I	Interrupts
P2.3	BREQ#	O	Bus controller
P2.4	EXTINT1	I	Interrupts
P2.5	HOLD#	I	Bus controller
P2.6	HLDA#	O	Bus controller
P2.7	CLKOUT	O	Clock generator
P3.0	CS0#	O	Chip-select unit
P3.1	CS1#	O	Chip-select unit
P3.2	CS2#	O	Chip-select unit
P3.3	CS3#	O	Chip-select unit
P3.4	CS4#	O	Chip-select unit
P3.5	CS5#	O	Chip-select unit
P3.6	EXTINT2	I	Interrupts
P3.7	EXTINT3	I	Interrupts
P4.0	PWM0	O	PWM
P4.1	PWM1	O	PWM
P4.2	PWM2	O	PWM
P4.3	—	I/O	—

Table 7-3 lists the registers associated with the bidirectional ports. Each port has three control registers (P*x*_MODE, P*x*_DIR, and P*x*_REG); they can be both read and written. The P*x*_PIN register is a status register that returns the logic level present on the pins; it can only be read. The registers are byte-addressable and can be windowed."Bidirectional Port Considerations" on page 7-9 discusses special considerations for reading P2_REG.7.

Table 7-3. Bidirectional Port Control and Status Registers

Mnemonic	Address	Description
P1_DIR P2_DIR P3_DIR P4_DIR	1FD2H 1FCBH 1FDAH 1FDBH	Port *x* Direction Each bit of P*x*_DIR controls the direction of the corresponding pin. 0 = complementary output (output only) 1 = input or open-drain output (input, output, or bidirectional) Open-drain outputs require external pull-ups.
P1_MODE P2_MODE P3_MODE P4_MODE	1FD0H 1FC9H 1FD8H 1FD9H	Port *x* Mode Each bit of P*x*_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. 0 = standard I/O port pin 1 = special-function signal
P1_PIN P2_PIN P3_PIN P4_PIN	1FD6H 1FCFH 1FDEH 1FDFH	Port *x* Input Each bit of P*x*_PIN reflects the current state of the corresponding pin, regardless of the pin configuration.
P1_REG P2_REG P3_REG P4_REG	1FD4H 1FCDH 1FDCH 1FDDH	Port *x* Data Output For an input, set the corresponding P*x*_REG bit. For an output, write the data to be driven out by each pin to the corresponding bit of P*x*_REG. When a pin is configured as standard I/O (P*x*_MODE.*y* = 0), the result of a CPU write to P*x*_REG is immediately visible on the pin. When a pin is configured as a special-function signal (P*x*_MODE.*y* = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to P*x*_REG, but the pin is unaffected until it is switched back to its standard I/O function. This feature allows software to configure a pin as standard I/O (clear P*x*_MODE.*y*), initialize or overwrite the pin value, then configure the pin as a special-function signal (set P*x*_MODE.*y*). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

7.2.1 Bidirectional Port Operation

Figure 7-1 shows the logic for driving the output transistors, Q1 and Q2. On ports 1, 2, and 3, Q1 can source at least –3 mA at V_{CC} – 0.7 volts. On port 4, which has a high-current sink capability for the PWMs, Q1 can source at least –3 mA at 0.45 volts. Q2 can sink at least 10 mA at 0.45 volts. (Consult the datasheet for specifications.)

In I/O mode (selected by clearing P*x*_MODE.*y)*, P*x*_REG and P*x*_DIR are input to the multiplexers. These signals combine to drive the gates of Q1 and Q2 so that the output is high, low, or high impedance. Table 7-4 is a logic table for I/O operation of these ports.

In special-function mode (selected by setting P*x*_MODE.*y*), SFDIR and SFDATA are input to the multiplexers. These signals combine to drive the gates of Q1 and Q2 so that the output is high, low, or high impedance. Special-function output signals clear SFDIR; special-function input signals set SFDIR. Table 7-5 is a logic table for special-function operation of these ports. Even if a pin is to be used in special-function mode, you must still initialize the pin as an input or output by writing to P*x*_DIR.

Resistor R1 provides ESD protection for the pin. Input signals are buffered. The ports use Schmitt-triggered buffers for improved noise immunity. The signals are latched into the P*x*_PIN sample latch and output onto the internal bus when the P*x*_PIN register is read.

The falling edge of RESET# turns on transistor Q3, which remains on for about 300 ns, causing the pin to change rapidly to its reset state. The active-low level of RESET# turns on transistor Q4, which weakly holds the pin high. (Q4 can source approximately –10 µA; consult the datasheet for exact specifications.) Q4 remains on, weakly holding the pin high, until your software writes to the P*x*_MODE register.

NOTE

P2.7 is an exception. After reset, P2.7 carries the CLKOUT signal rather than being held high. When CLKOUT is selected, it is always a complementary output.

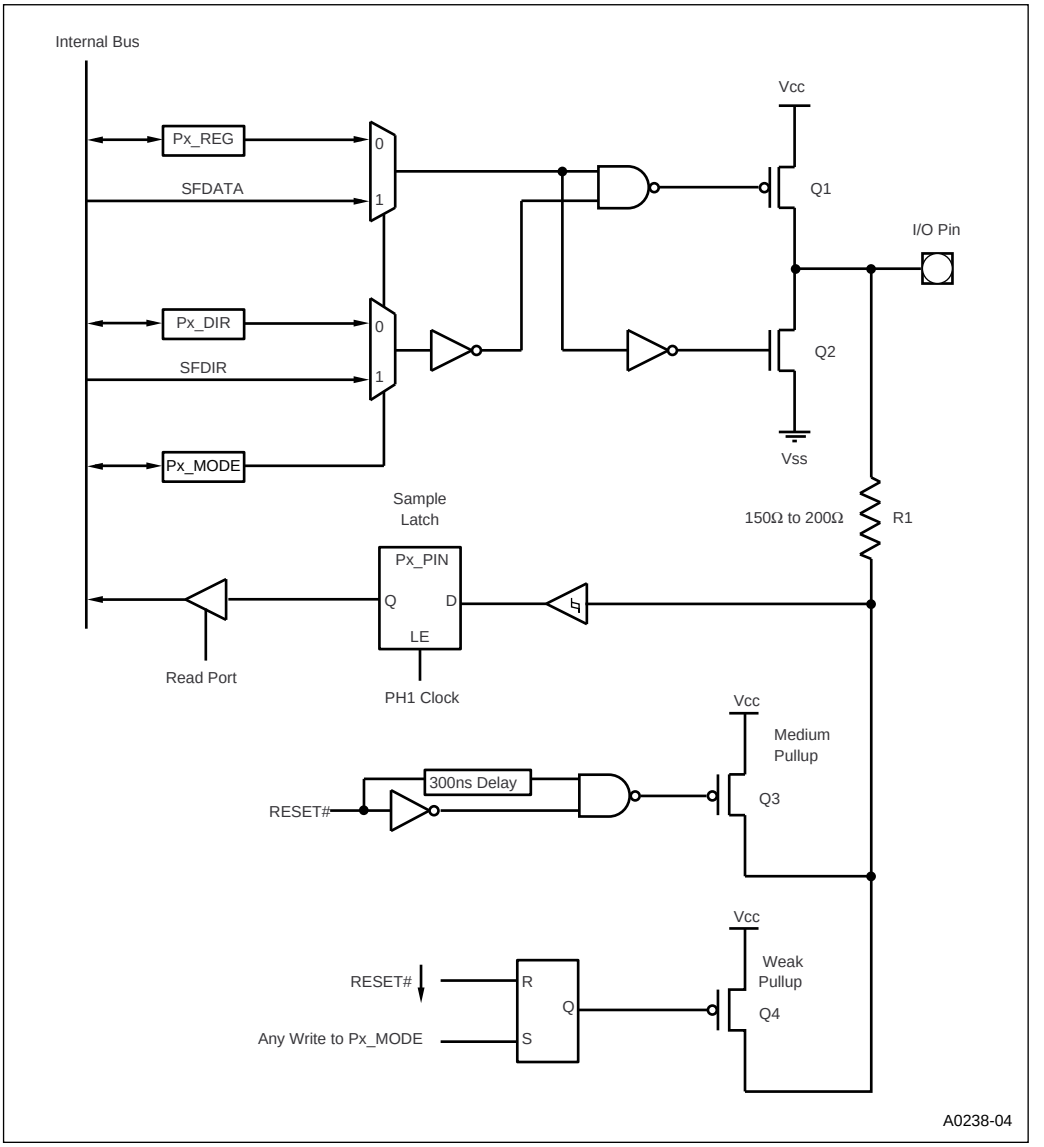


Figure 7-1. Bidirectional Port Structure

Table 7-4. Logic Table for Bidirectional Ports in I/O Mode

Configuration	Complementary Output		Open-drain Output	Input
P*x*_MODE	0	0	0	0
P*x*_DIR	0	0	1	1
SFDIR	X	X	X	X
SFDATA	X	X	X	X
P*x*_REG	0	1	0, 1 (Note 2)	1
Q1	off	on	off	off
Q2	on	off	on, off (Note 2)	off
P*x*_PIN	0	1	X (Note 3)	high-impedance (Note 4)

NOTES:
1. X = Don't care.
2. If P*x*_REG is cleared, Q2 is on; if P*x*_REG is set, Q2 is off.
3. P*x*_PIN contains the current value on the pin.
4. During reset and until the first write to P*x*_MODE, Q4 is on.

Table 7-5. Logic Table for Bidirectional Ports in Special-function Mode

Configuration	Complementary Output		Open-drain Output	Input
P*x*_MODE	1	1	1	1
P*x*_DIR	0	0	1	1
SFDIR	0	0	1	1
SFDATA	0	1	0, 1 (Note 2)	1
P*x*_REG	X	X	X	1
Q1	off	on	off	off
Q2	on	off	on, off (Note 2)	off
P*x*_PIN	0	1	X (Note 3)	high-impedance (Note 4)

NOTES:
1. X = Don't care.
2. If P*x*_REG is cleared, Q2 is on; if P*x*_REG is set, Q2 is off.
3. P*x*_PIN contains the current value on the pin.
4. During reset and until the first write to P*x*_MODE, Q4 is on.

7.2.2 Bidirectional Port Pin Configurations

Each bidirectional port pin can be individually configured to operate either as an I/O pin or as a pin for a special-function signal. In the special-function configuration, the signal is controlled by an on-chip peripheral or an off-chip component. In either configuration, two modes are possible:

- complementary output (output only)
- high-impedance input or open-drain output (input, output, or bidirectional)

To prevent the CMOS inputs from floating, the bidirectional port pins are weakly pulled high during and after reset, until your software writes to P*x*_MODE. The default values of the control registers after reset configure the pins as high-impedance inputs with weak pull-ups. To ensure that the ports are initialized correctly and that the weak pull-ups are turned off, follow this suggested initialization sequence:

1. Write to P*x*_DIR to establish the individual pins as either inputs or outputs. (Outputs will drive the data that you specify in step 3.)
   — For a complementary output, clear its P*x*_DIR bit.
   — For a high-impedance input or an open-drain output, set its P*x*_DIR bit. (Open-drain outputs require external pull-ups.)
2. Write to P*x*_MODE to select either I/O or special-function mode. Writing to P*x*_MODE (regardless of the value written) turns off the weak pull-ups. Even if the entire port is to be used as I/O (its default configuration after reset), **you must write to P*x*_MODE to ensure that the weak pull-ups are turned off**.
   — For a standard I/O pin, clear its P*x*_MODE bit. In this mode, the pin is driven as defined in steps 1 and 3.
   — For a special-function signal, set its P*x*_MODE bit. In this mode, the associated peripheral controls the pin.
3. Write to P*x*_REG.
   — For output pins defined in step 1, write the data that is to be driven by the pins to the corresponding P*x*_REG bits. For special-function outputs, the value is immaterial because the peripheral controls the pin. However, you must still write to P*x*_REG to initialize the pin.
   — For input pins defined in step 1, set the corresponding P*x*_REG bits.

Table 7-6 lists the control register values for each possible configuration. For special-function outputs, the P*x*_REG value is irrelevant (don't care) because the associated peripheral controls the pin in special-function mode. However, you must still write to P*x*_REG to initialize the pin. For a bidirectional pin to function as an input (either special function or port pin), you must set P*x*_REG.

Table 7-6. Control Register Values for Each Configuration

Desired Pin Configuration	Configuration Register Settings		
Standard I/O Signal	**P*x*_DIR**	**P*x*_MODE†**	**P*x*_REG**
Complementary output, driving 0	0	0	0
Complementary output, driving 1	0	0	1
Open-drain output, strongly driving 0	1	0	0
Open-drain output, high impedance	1	0	1
Input	1	0	1
Special-function signal	**P*x*_DIR**	**P*x*_MODE†**	**P*x*_REG**
Complementary output, output value controlled by peripheral	0	1	X
Open-drain output, output value controlled by peripheral	1	1	X
Input	1	1	1

† During reset and until the first write to P*x*_MODE, the pins are weakly held high.

7.2.3 Bidirectional Port Pin Configuration Example

Assume that you wish to configure the pins of a bidirectional port as shown in Table 7-7.

Table 7-7. Port Configuration Example

Port Pin(s)	Configuration	Data
P*x*.0, P*x*.1	high-impedance input	high-impedance
P*x*.2, P*x*.3	open-drain output	0
P*x*.4	open-drain output	1 (assuming external pull-up)
P*x*.5, P*x*.6	complementary output	0
P*x*.7	complementary output	1

To do so, you could use the following example code segment. Table 7-8 shows the state of each pin after reset and after execution of each line of the example code.

```
LDB   Px_DIR,#00011111B
LDB   Px_MODE,#00000000B
LDB   Px_REG,#10010011B
```

Table 7-8. Port Pin States After Reset and After Example Code Execution

Action or Code	Resulting Pin States†							
	P*x*.7	P*x*.6	P*x*.5	P*x*.4	P*x*.3	P*x*.2	P*x*.1	P*x*.0
Reset	wk1	wk1	wk1	wk1	wk1	wk1	wk1	wk1
LDB P*x*_DIR, #00011111B	1	1	1	wk1	wk1	wk1	wk1	wk1
LDB P*x*_MODE, #00000000B	1	1	1	HZ1	HZ1	HZ1	HZ1	HZ1
LDB P*x*_REG, #10010011B	1	0	0	HZ1	0	0	HZ1	HZ1

† wk1 = weakly pulled high, HZ1 = high impedance (actually a "1" with an external pull-up).

7.2.4 Bidirectional Port Considerations

This section outlines special considerations for using the pins of these ports.

Port 1 — After reset, your software must configure the device to match the external system. This is accomplished by writing appropriate configuration data into P1_MODE. Writing to P1_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 7-1 on page 7-5). For this reason, even if port 1 is to be used as it is configured at reset, you should still write data into P1_MODE.

Port 2 — After reset, your software must configure the device to match the external system. This is accomplished by writing appropriate configuration data into P2_MODE. Writing to P2_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 7-1 on page 7-5). For this reason, even if port 2 is to be used as it is configured at reset, you should still write data into P2_MODE.

P2.2/EXTINT0 — Writing to P2_MODE.2 sets the EXTINT0 interrupt pending bit (INT_PEND.3). After configuring the port pins, clear the interrupt pending registers before globally enabling interrupts. See "Design Considerations for External Interrupt Inputs" on page 7-11.

P2.4/EXTINT1 — Writing to P2_MODE.4 sets the EXTINT1 interrupt pending bit (INT_PEND.4). After configuring the port pins, clear the interrupt pending registers before globally enabling interrupts. See "Design Considerations for External Interrupt Inputs" on page 7-11.

P2.5/HOLD# — If P2.5 is configured as a standard I/O port pin, the device does not recognize signals on this pin as HOLD#. Instead, the bus controller receives an internal HOLD signal. This enables the device to access the external bus while it is performing I/O at P2.5.

P2.7/CLKOUT	Following reset, P2.7 carries the strongly driven CLKOUT signal. It is **not** held high. When P2.7 is configured as CLKOUT, it is always a complementary output.
P2.7	A value written to P2_REG.7 is held in a buffer until P2_MODE.7 is cleared, at which time the value is loaded into P2_REG.7. A value read from P2_REG.7 is the value currently in the register, not the value in the buffer. Therefore, any change to P2_REG.7 can be read only after P2_MODE.7 is cleared.
Port 3	After reset, your software must configure the device to match the external system. This is accomplished by writing appropriate configuration data into P3_MODE. Writing to P3_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 7-1 on page 7-5). For this reason, even if port 3 is to be used as it is configured at reset, you should still write data into P3_MODE.
P3.0/CS0#	P3.0/CS0# is weakly pulled high during reset. After reset, it defaults to the CS0# function. This chip-select signal detects address ranges that contain the CCBs and FF2080H (program start-up address). See Chapter 13, "Interfacing with External Memory," for a detailed description of chip-select signal functions after reset.
P3.6/EXTINT2	Writing to P3_MODE.6 sets the EXTINT2 interrupt pending bit (INT_PEND1.5). After configuring the port pins, clear the interrupt pending registers before globally enabling interrupts. See "Design Considerations for External Interrupt Inputs" on page 7-11.
P3.7/EXTINT3	Writing to P3_MODE.7 sets the EXTINT3 interrupt pending bit (INT_PEND1.6). After configuring the port pins, clear the interrupt pending registers before globally enabling interrupts. See "Design Considerations for External Interrupt Inputs" on page 7-11.
Port 4	After reset, your software must configure the device to match the external system. This is accomplished by writing appropriate configuration data into P4_MODE. Writing to P4_MODE not only configures the pins but also turns off the transistor that weakly holds the pins high (Q4 in Figure 7-1 on page 7-5). For this reason, even if port 4 is to be used as it is configured at reset, you should still write data into P4_MODE.

7.2.5 Design Considerations for External Interrupt Inputs

To configure a port pin that serves as an external interrupt input, you must set the corresponding bits in the configuration registers (P*x*_DIR, P*x*_MODE, and P*x*_REG). However, setting the P*x*_MODE bit causes the device to set the corresponding interrupt pending bit, indicating an interrupt request. To configure P2.2/EXTINT0, P2.4/EXTINT1, P3.6/EXTINT2, and P3.7/EXTINT3, we recommend the following sequence to prevent the false interrupt request:

1. Disable interrupts by executing the DI instruction.
2. Set the P*x*_DIR bit.
3. Set the P*x*_MODE bit.
4. Set the P*x*_REG bit.
5. Clear the INT_PEND and INT_PEND1 bits.
6. Enable interrupts (optional) by executing the EI instruction.

7.3 EPORT

The EPORT is a four-bit, bidirectional, memory-mapped I/O port in the 8XC196NP, but a standard I/O port in the 80C196NU. For the 8XC196NP, it must be accessed using indirect or indexed addressing, and it cannot be windowed. For the 80C196NU, it can be windowed. This port provides the address signals necessary to support extended addressing. If one or more extended address pins are unnecessary in an application, the unused port pins can be used for I/O. Figure 7-2 shows a block diagram of the EPORT.

Table 7-9 lists the EPORT pins with their extended-address signals. Table 7-10 lists the registers that affect the function and indicate the status of EPORT pins.

Table 7-9. EPORT Pins

Port Pin	Extended-address Signal	Signal Type
EPORT.0	A16	I/O
EPORT.1	A17	I/O
EPORT.2	A18	I/O
EPORT.3	A19	I/O

Table 7-10. EPORT Control and Status Registers

Mnemonic	Address	Description
EP_DIR	1FE3H	EPORT Direction In I/O mode, each bit of EP_DIR controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as either an input or an open-drain output. (Open-drain outputs require external pull-ups). Any pin that is configured for its extended-address function is forced to the complementary output mode except during reset, hold, idle, powerdown, and standby. (Standby mode is available only on the 80C196NU.)
EP_MODE	1FE1H	EPORT Mode Each bit of EP_MODE controls whether the corresponding pin functions as a standard I/O port pin or as an extended-address signal. Setting a bit configures a pin as an extended-address signal; clearing a bit configures a pin as a standard I/O port pin.
EP_PIN	1FE7H	EPORT Pin State Each bit of EP_PIN reflects the current state of the corresponding pin, regardless of the pin configuration.
EP_REG	1FE5H	EPORT Data Output Each bit of EP_REG contains data to be driven out by the corresponding pin. When a pin is configured as standard I/O (EP_MODE.*x* = 0), the result of a CPU write to EP_REG is immediately visible on the pin. During nonextended data accesses, EP_REG contains the value of the memory page that is to be accessed. For compatibility with software tools, clear the EP_REG bit for any EPORT pin that is configured as an extended-address signal (EP_MODE.*x* set). **80C196NU Only:** For nonextended data accesses, the 80C196NU forces the page address to 00H. You cannot change pages by modifying EP_REG.

7.3.1 EPORT Operation

As Figure 7-2 shows, each EPORT pin serves either as I/O or as an address line, as selected by the I/O multiplexer. This multiplexer is controlled by the EP_MODE register. If EP_MODE.*x* is clear (I/O mode), the pin serves as I/O until EP_MODE.*x* is changed.

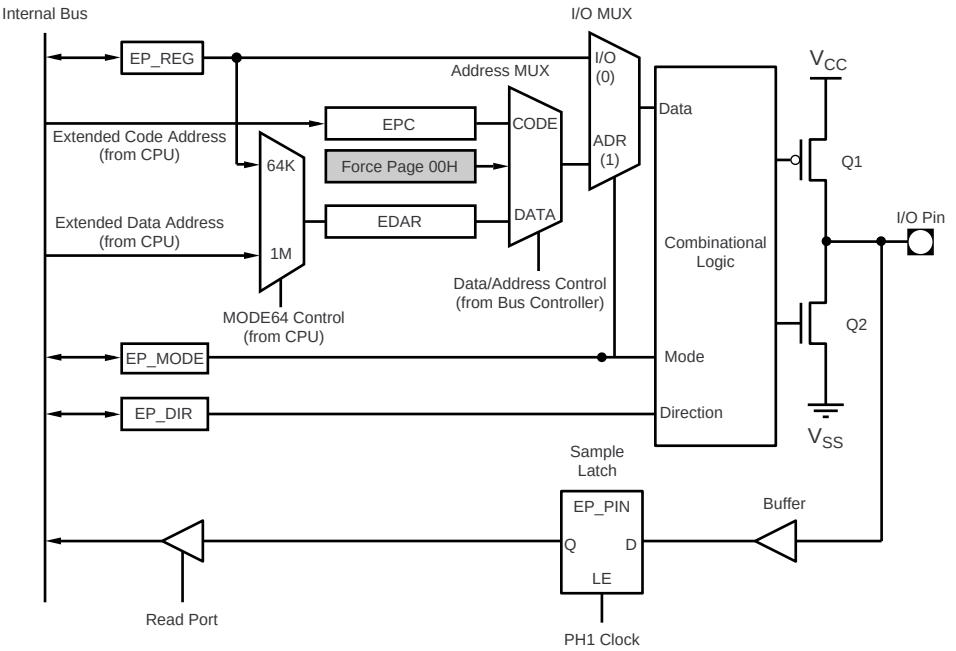


Figure 7-2. EPORT Block Diagram

If EP_MODE.*x* is set (address mode), the address multiplexer determines the address source. For an instruction fetch, the address multiplexer is set to the CODE input, which selects the extended program counter (EPC) as the address source. For a data fetch, or when there is no external bus activity, the address multiplexer is set to the DATA input, which selects the extended data address register (EDAR) as the address source.

The EDAR is loaded from two different sources, depending on whether the data access is extended or nonextended. For extended data accesses, the data multiplexer is set to the 1-Mbyte mode input and EDAR is loaded with the extended address. For nonextended data accesses, the data multiplexer is set to the 64-Kbyte mode input and EDAR is loaded from EP_REG. The last value loaded remains in EDAR until the next data access. (Refer to “Fetching Code and Data in the 1-Mbyte and 64-Kbyte Modes” on page 5-23 for more information.)

The 8XC196NP allows you to change the value of EP_REG to control which memory page a non-extended instruction accesses. However, software tools require that EP_REG be equal to 00H. The 80C196NU forces all nonextended data accesses to page 00H. You cannot use EP_REG to change pages.

You can read EP_PIN at any time to determine the value of a pin. When EP_PIN is read, the contents of the sample latch are output onto the internal bus.

Figure 7-3 shows a circuit schematic for a single bit of the EPORT. Q1 and Q2 are the strong complementary drivers for the pin. Q1 can source at least –3 mA at V_{CC} – 0.7 volts. Q2 can sink at least 3 mA at V_{SS} + 0.45 volts. (Consult the datasheet for specifications.) Resistor R1 provides ESD protection for the pin.

7.3.1.1 Reset

During reset, the falling edge of RESET# generates a short pulse that turns on the medium pull-up transistor Q3, which remains on for about 300 ns, causing the pin to change rapidly to its reset state. The active-low level of RESET# turns on transistor Q4, which weakly holds the pin high. (Q4 can source approximately –10 µA; consult the datasheet for exact specifications.) When RESET# is inactive, both Q3 and Q4 are off; Q1 and Q2 determine output drive.

7.3.1.2 Output Enable

If RESET#, HOLD#, idle, or powerdown is asserted, the gates that control Q1 and Q2 are disabled and Q1 and Q2 remain off. Otherwise, the gates are enabled and complementary or open-drain operation is possible.

7.3.1.3 Complementary Output Mode

For complementary output mode, the gates that control Q1 and Q2 must be enabled. The Q2 gate is always enabled (except when RESET#, HOLD#, idle, or powerdown is asserted). Either clearing EP_DIR (selecting complementary mode) or setting EP_MODE (selecting address mode) enables the logic gate preceding Q1. The value of DATA determines which transistor is turned on. If DATA is equal to one, Q1 is turned on and the pin is pulled high. If DATA is equal to zero, Q2 is turned on and the pin is pulled low.

7.3.1.4 Open-drain Output Mode

For open-drain output mode, the gate that controls Q1 must be disabled. Setting EP_DIR (selecting open-drain mode) **and** clearing EP_MODE (selecting I/O mode) disables the logic gate preceding Q1. The value of DATA determines whether Q2 is turned on. If DATA is equal to one, both Q1 and Q2 remain off and the pin is left in high-impedance state (floating). If DATA is equal to zero, Q2 is turned on and the pin is pulled low.

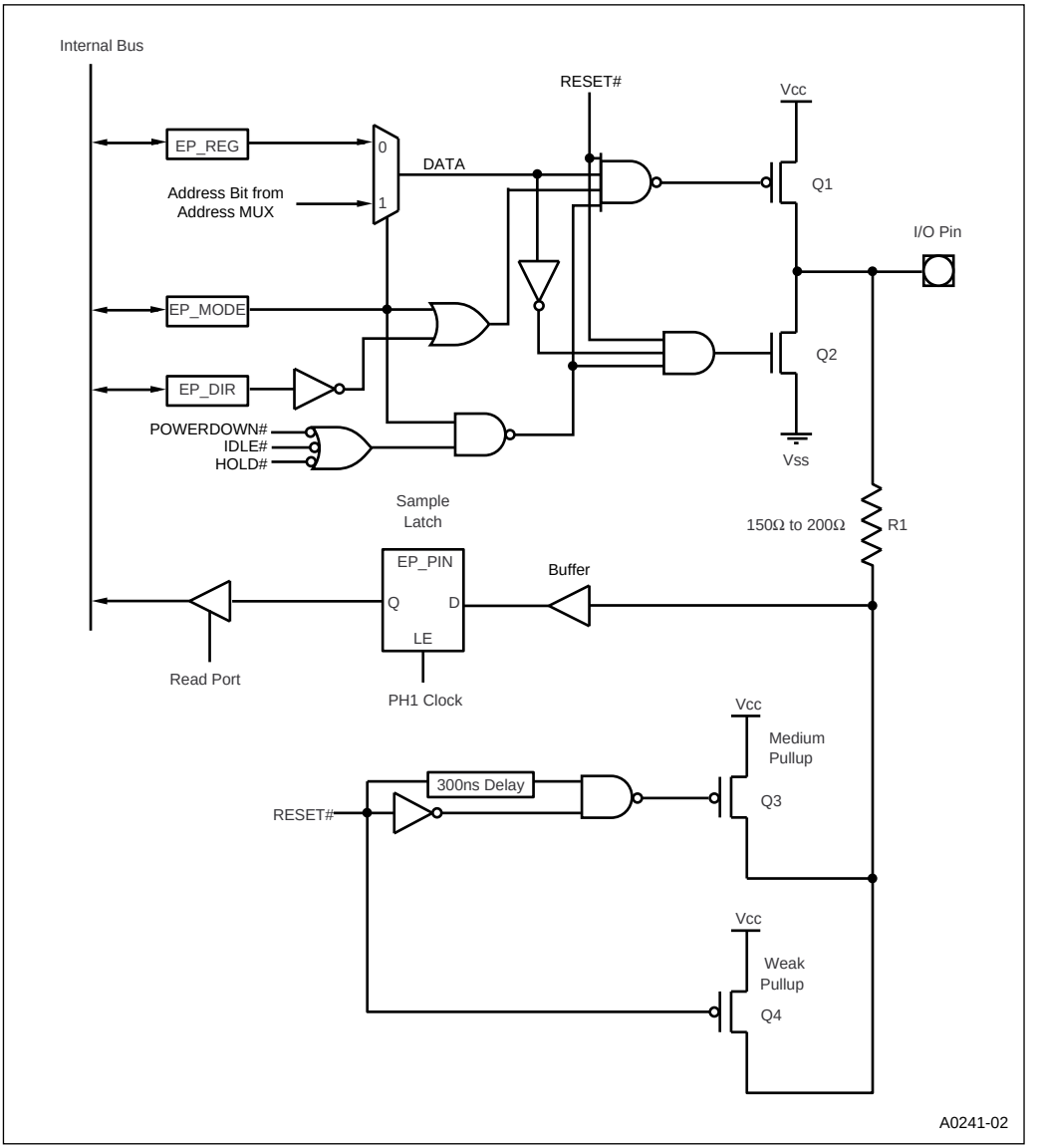


Figure 7-3. EPORT Structure

7.3.1.5 Input Mode

Input mode is obtained by configuring the pin as an open-drain output (EP_DIR set and EP_MODE clear) and writing a one to EP_REG.*x*. In this configuration, Q1 and Q2 are both off, allowing an external device to drive the pin. To determine the value of the I/O pin, read EP_PIN.*x*.

Table 7-11 is a logic table for I/O operation and Table 7-12 is a logic table for address mode operation of EPORT.

Table 7-11. Logic Table for EPORT in I/O Mode

Configuration	Complementary Output		Open-drain Output	Input
EP_MODE	0	0	0	0
EP_DIR	0	0	0, 1 (Note 2)	1
EP_REG	0	1	0	1
Address Bit	X	X	X	X
Q1	off	on	off	off
Q2	on	off	on	off
EP_PIN	0	1	0	high-impedance

NOTES:
1. X = Don't care.
2. If EP_REG is clear, Q2 is on; if EP_REG is set, Q2 is off.

Table 7-12. Logic Table for EPORT in Address Mode

Configuration	Complementary Output (Note 1)	
EP_MODE	1	1
EP_DIR	X	X
EP_REG	X (Note 2)	X (Note 2)
Address Bit	0	1
Q1	off	on
Q2	on	off
EP_PIN	0	1

NOTES:
1. X = Don't care.
2. EP_REG is output on EPORT during any nonextended external memory access.

7.3.2 Configuring EPORT Pins

Each EPORT pin can be individually configured to operate either as an extended-address signal or as an I/O pin in one of these modes:

- complementary output (output only)
- high-impedance input or open-drain output (input, output, or bidirectional)

7.3.2.1 Configuring EPORT Pins for Extended-address Functions

The EPORT pins default to their extended-address functions upon reset (see Table B-5 on page B-13). During program execution, the pins can be reconfigured at any time from address to I/O and back to address. However, this is not recommended unless you understand the implications of changing memory addressing "on the fly." To change a pin from I/O to address, clear the EP_REG.*x* bit and set the EP_MODE.*x* bit. (Clearing EP_REG.*x* is required for compatibility with software development tools.)

7.3.2.2 Configuring EPORT Pins for I/O

To configure a pin for I/O, write the appropriate values to the control registers, in this order:

1. EP_DIR
2. EP_MODE
3. EP_REG

Table 7-13 lists the register settings for the EPORT pins.

Table 7-13. Configuration Register Settings for EPORT Pins

Desired Pin Configuration	Configuration Register Settings			EP_PIN Value
	EP_DIR	EP_MODE	EP_REG	
Address	X†	1	0††	address
Complementary output	0	0	data value	data value
Open-drain output	1	0	data value	data value
Input	1	0	1	I/O pin value

† X = Don't care.
†† Must be zero for compatibility with software tools.

7.3.3 EPORT Considerations

This section outlines considerations for using the EPORT pins.

7.3.3.1 EPORT Status During Reset, CCB Fetch, Idle, Powerdown, and Hold

During reset, the EPORT pins are forced to their extended-address functions and are weakly pulled high. During the CCB fetch, FFH is strongly driven onto the pins. This value remains strongly driven until either the pin is configured for I/O or a different extended address is accessed. If the pins remain configured as extended-address functions, they are placed in a high-impedance state during idle, powerdown, standby (80C196NU only), and hold. If they are configured as I/O, they retain their I/O function during those modes. See Figure 11-7 on page 11-8 and Table B-5 on page B-13 for additional information.

7.3.3.2 EP_REG Settings for Pins Configured as Extended-address Signals

Nonextended data accesses go to the address contained in EP_REG. Therefore, if you configure EP_REG to point to the desired address, you can use nonextended addressing modes to access the extended address space. However, we recommend that you clear the EP_REG bits for any EPORT pins configured as extended-address signals in order to maintain compatibility with software development tools.

NOTE

If any pins are configured as extended-address signals and their corresponding EP_REG bits are set, nonextended operations will still access the register file and standard SFRs. However, all other nonextended accesses, including those to internal RAM and internal nonvolatile memory, will be directed off-chip to the "page" address in EP_REG.

The 8XC196NP allows you to change the value of EP_REG to control which memory page a nonextended instruction accesses. However, software tools require that EP_REG be equal to 00H. The 80C196NU forces all nonextended data accesses to page 00H. You cannot use EP_REG to change pages.

7.3.3.3 EPORT Status During Instruction Execution

When using the EPORT to address memory outside page 00H, keep these points in mind:

1. During extended accesses, the upper four bits of the address (lower four bits of the EPC) are sent to the EPORT. EPORT pins configured for the extended-address function (EP_MODE.*x* set) output this address.
2. During nonextended accesses, EPORT pins configured for the extended-address function (EP_MODE.*x* set) output the value contained in EP_REG.

3. Any nonextended or direct instruction that accesses the register file or the windowable SFRs is always directed internally to these areas, regardless of the page from which code is executing. This effectively maps the register file and windowable SFRs into every page. Extended instructions can access the “mapped over” areas of each page, as shown in the following code example.

```
EST  1CH, 01001CH[0]     ;reg 1CH stored at memory location 01001CH
```

7.3.3.4 Design Considerations

At the end of EPORT bus activity and during periods of internal bus activity, EPORT pins continue to drive the last data address that was output. If these lines are being used to enable external memory, that memory will remain enabled until a different page is accessed.

During the CCB fetch, all EPORT lines are strongly driven high. Designers should ensure that this does not conflict with external systems that are outputting signals to the EPORT.

When EPORT pins are floated during idle, powerdown, or hold, the external system must provide circuitry to prevent CMOS inputs on **external** devices from floating. During powerdown, the EPORT input buffers on pins configured for their extended-address function are disconnected from the pins, so a floating pin will not cause increased power consumption.

Open-drain outputs require an external pull-up resistor. Inputs must be driven or pulled high or low; they must **not** be allowed to float.

8

Serial I/O (SIO) Port

CHAPTER 8
SERIAL I/O (SIO) PORT

A serial input/output (SIO) port provides a means for the system to communicate with external devices. This device has a serial I/O (SIO) port that shares pins with port 2. This chapter describes the SIO port and explains how to configure it. Chapter 7, “I/O Ports,” explains how to configure the port pins for their special functions. Refer to Appendix B for details about the signals discussed in this chapter.

8.1 SERIAL I/O (SIO) PORT FUNCTIONAL OVERVIEW

The serial I/O port (Figure 8-1) is an asynchronous/synchronous port that includes a universal asynchronous receiver and transmitter (UART). The UART has one synchronous mode (mode 0) and three asynchronous modes (modes 1, 2, and 3) for both transmission and reception.

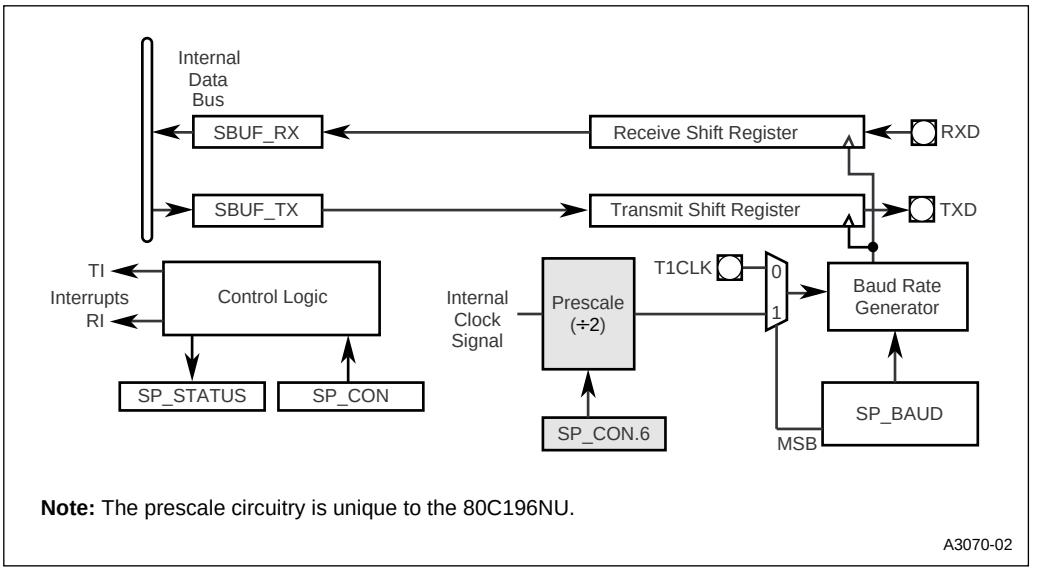


Figure 8-1. SIO Block Diagram

The serial port receives data into the receive buffer; it transmits data from the port through the transmit buffer. The transmit and receive buffers are separate registers, permitting simultaneous reads and writes to both. The transmitter and receiver are buffered to support continuous transmissions and to allow reception of a second byte before the first byte has been read.

An independent, 15-bit baud-rate generator controls the baud rate of the serial port. Either the internal peripheral clock or T1CLK can provide the clock signal. The baud-rate register (SP_BAUD) selects the clock source and the baud rate.

8.2 SERIAL I/O PORT SIGNALS AND REGISTERS

Table 8-1 describes the SIO signals and Table 8-2 describes the control and status registers.

Table 8-1. Serial Port Signals

Port Pin	Serial Port Signal	Serial Port Signal Type	Description
P2.0	TXD	O	Transmit Serial Data In modes 1, 2, and 3, TXD transmits serial port output data. In mode 0, it is the serial clock output.
P2.1	RXD	I/O	Receive Serial Data In modes 1, 2, and 3, RXD receives serial port input data. In mode 0, it functions as an input or an open-drain output for data.
P1.4	T1CLK	I	Timer 1 Clock External clock source for the baud-rate generator input.

Table 8-2. Serial Port Control and Status Registers

Mnemonic	Address	Description
INT_MASK	0013H	Interrupt Mask Setting the TI bit enables the transmit interrupt; clearing the bit disables (masks) the interrupt. Setting the RI bit enables the receive interrupt; clearing the bit disables (masks) the interrupt.
INT_PEND	0012H	Interrupt Pending When set, the TI bit indicates a pending transmit interrupt. When set, the RI bit indicates a pending receive interrupt.
P1_DIR	1FD2H	Port 1 Direction This register selects the direction of each port 1 pin. To use T1CLK as the input clock to the baud-rate generator, clear P1_DIR.4.
P1_MODE	1FD0H	Port 1 Mode This register selects either the general-purpose input/output function or the peripheral function for each pin of port 1. To use T1CLK as the clock source for the baud-rate generator, set P1_MODE.4 to configure T1CLK (P1.4) for the SIO port.

Table 8-2. Serial Port Control and Status Registers (Continued)

Mnemonic	Address	Description
P1_PIN	1FD6H	Port 1 Pin State If you are using T1CLK (P1.4) as the clock source for the baud-rate generator, you can read P1_PIN.4 to determine the current value of T1CLK.
P1_REG	1FD4H	Port 1 Output Data To use T1CLK as the clock source for the baud-rate generator, set P1_REG.4.
P2_DIR	1FCBH	Port 2 Direction This register selects the direction of each port 2 pin. Clear P2_DIR.1 to configure RXD (P2.1) as a high-impedance input/open-drain output, and set P2_DIR.0 to configure TXD (P2.0) as a complementary output.
P2_MODE	1FC9H	Port 2 Mode This register selects either the general-purpose input/output function or the peripheral function for each pin of port 2. Set P2_MODE.1:0 to configure TXD (P2.0) and RXD (P2.1) for the SIO port.
P2_PIN	1FCFH	Port 2 Pin State Two bits of this register contain the values of the TXD (P2.0) and RXD (P2.1) pins. Read P2_PIN to determine the current value of the pins.
P2_REG	1FCDH	Port 2 Output Data This register holds data to be driven out on the pins of port 2. Set P2_REG.1 for the RXD (P2.1) pin. Write the desired output data for the TXD (P2.0) pin to P2_REG.0.
SBUF_RX	1FB8H	Serial Port Receive Buffer This register contains data received from the serial port.
SBUF_TX	1FBAH	Serial Port Transmit Buffer This register contains data that is ready for transmission. In modes 1, 2, and 3, writing to SBUF_TX starts a transmission. In mode 0, writing to SBUF_TX starts a transmission only if the receiver is disabled (SP_CON.3 = 0)
SP_BAUD	1FBCH,1FBDH	Serial Port Baud Rate This register selects the serial port baud rate and clock source. The most-significant bit selects the clock source. The lower 15 bits represent the BAUD_VALUE, an unsigned integer that determines the baud rate.
SP_CON	1FBBH	Serial Port Control This register selects the communications mode and enables or disables the receiver, parity checking, and ninth-bit data transmissions. The TB8 bit is cleared after each transmission.

Table 8-2. Serial Port Control and Status Registers (Continued)

Mnemonic	Address	Description
SP_STATUS	1FB9H	Serial Port Status This register contains the serial port status bits. It has status bits for receive overrun errors (OE), transmit buffer empty (TXE), framing errors (FE), transmit interrupt (TI), receive interrupt (RI), and received parity error (RPE) or received bit 8 (RB8). Reading SP_STATUS clears all bits except TXE; writing a byte to SBUF_TX clears the TXE bit.

8.3 SERIAL PORT MODES

The serial port has both synchronous and asynchronous operating modes for transmission and reception. This section describes the operation of each mode.

8.3.1 Synchronous Mode (Mode 0)

The most common use of mode 0, the synchronous mode, is to expand the I/O capability of the device with shift registers (see Figure 8-2). In this mode, the TXD pin outputs a set of eight clock pulses, while the RXD pin either transmits or receives data. Data is transferred eight bits at a time with the least-significant bit first. Figure 8-3 shows a diagram of the relative timing of these signals. Note that only mode 0 uses RXD as an open-drain output.

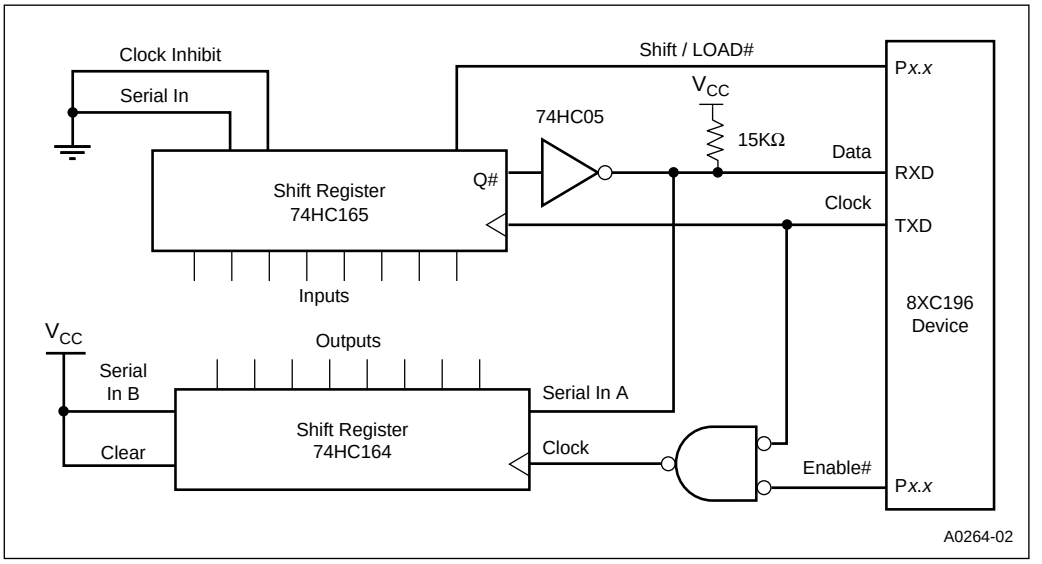


Figure 8-2. Typical Shift Register Circuit for Mode 0

In mode 0, RXD must be enabled for receptions and disabled for transmissions. (See “Programming the Control Register” on page 8-8.) When RXD is enabled, either a rising edge on the RXD input or clearing the receive interrupt (RI) flag in SP_STATUS starts a reception. When RXD is disabled, writing to SBUF_TX starts a transmission.

Disabling RXD stops a reception in progress and inhibits further receptions. To avoid a partial or undesired complete reception, disable RXD before clearing the RI flag in SP_STATUS. This can be handled in an interrupt environment by using software flags or in straight-line code by using the interrupt pending register to signal the completion of a reception.

During a reception, the RI flag in SP_STATUS is set after the stop bit is sampled. The RI pending bit in the interrupt pending register is set immediately before the RI flag is set. During a transmission, the TI flag is set immediately after the end of the last (eighth) data bit is transmitted. The TI pending bit in the interrupt pending register is generated when the TI flag in SP_STATUS is set.

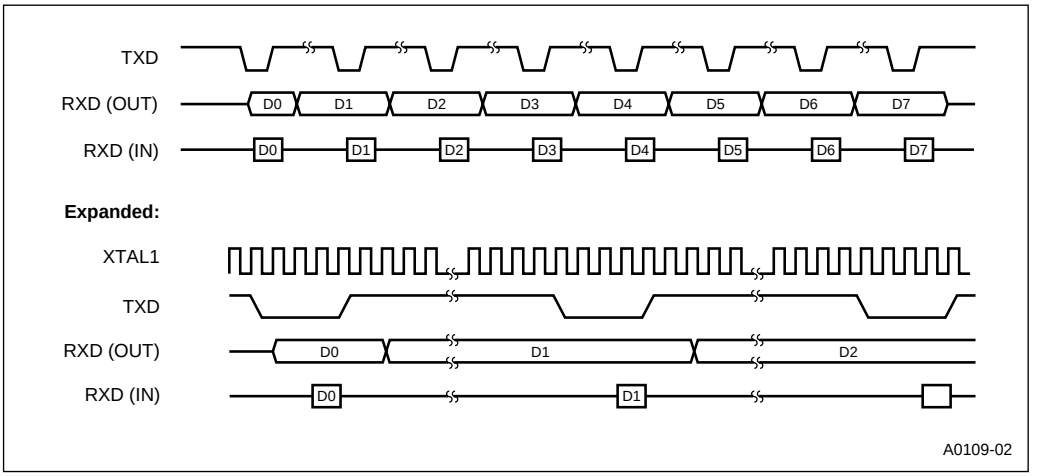


Figure 8-3. Mode 0 Timing

8.3.2 Asynchronous Modes (Modes 1, 2, and 3)

Modes 1, 2, and 3 are full-duplex serial transmit/receive modes, meaning that they can transmit and receive data simultaneously. Mode 1 is the standard 8-bit, asynchronous mode used for normal serial communications. Modes 2 and 3 are 9-bit asynchronous modes typically used for interprocessor communications (see “Multiprocessor Communications” on page 8-8). In mode 2, the serial port sets an interrupt pending bit only if the ninth data bit is set. In mode 3, the serial port always sets an interrupt pending bit upon completion of a data transmission or reception.

When the serial port is configured for mode 1, 2, or 3, writing to SBUF_TX causes the serial port to start transmitting data. New data placed in SBUF_TX is transmitted only after the stop bit of the previous data has been sent. A falling edge on the RXD input causes the serial port to begin receiving data if RXD is enabled. Disabling RXD stops a reception in progress and inhibits further receptions. (See "Programming the Control Register" on page 8-8.)

8.3.2.1 Mode 1

Mode 1 is the standard asynchronous communications mode. The data frame used in this mode (Figure 8-4) consists of ten bits: a start bit (0), eight data bits (LSB first), and a stop bit (1). If parity is enabled, a parity bit is sent instead of the eighth data bit, and parity is checked on reception.

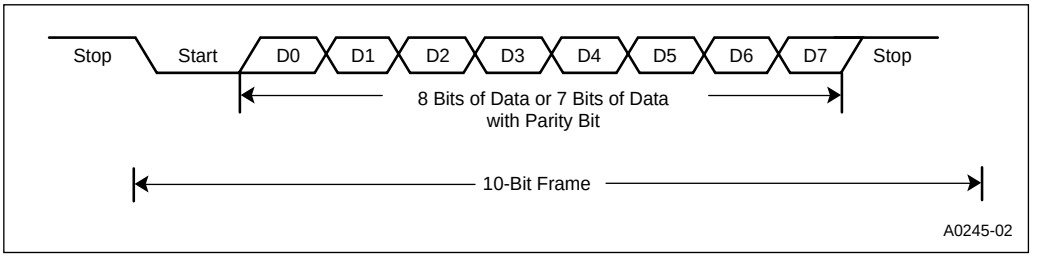


Figure 8-4. Serial Port Frames for Mode 1

The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud-rate generator is initialized. The receive shift clock is reset when a start bit (high-to-low transition) is received. Therefore, the transmit clock may not be synchronized with the receive clock, although both will be at the same frequency.

The transmit interrupt (TI) and receive interrupt (RI) flags in SP_STATUS are set to indicate completed operations. During a reception, both the RI flag and the RI interrupt pending bit are set just before the end of the stop bit. During a transmission, both the TI flag and the TI interrupt pending bit are set at the beginning of the stop bit. The next byte cannot be sent until the stop bit is sent.

Use caution when connecting more than two devices with the serial port in half-duplex (i.e., with one wire for transmit and receive). The receiving processor must wait for one bit time after the RI flag is set before starting to transmit. Otherwise, the transmission could corrupt the stop bit, causing a problem for other devices listening on the link.

8.3.2.2 Mode 2

Mode 2 is the asynchronous, ninth-bit recognition mode. This mode is commonly used with mode 3 for multiprocessor communications. Figure 8-5 shows the data frame used in this mode. It consists of a start bit (0), nine data bits (LSB first), and a stop bit (1). During transmissions, setting the TB8 bit in the SP_CON register before writing to SBUF_TX sets the ninth transmission bit. The hardware clears the TB8 bit after every transmission, so it must be set (if desired) before each write to SBUF_TX. During receptions, the RI flag and RI interrupt pending bit are set only if the TB8 bit is set. This provides an easy way to have selective reception on a data link. (See "Multiprocessor Communications" on page 8-8). Parity cannot be enabled in this mode.

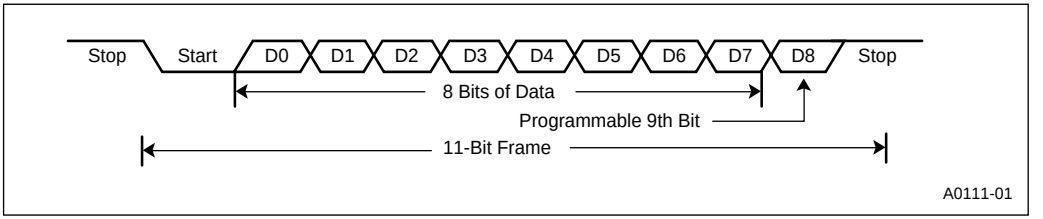


Figure 8-5. Serial Port Frames in Mode 2 and 3

8.3.2.3 Mode 3

Mode 3 is the asynchronous, ninth-bit mode. The data frame for this mode is identical to that of mode 2. Mode 3 differs from mode 2 during transmissions in that parity can be enabled, in which case the ninth bit becomes the parity bit. When parity is disabled, data bits 0–7 are written to the serial port transmit buffer, and the ninth data bit is written to bit 4 (TB8) bit in the SP_CON register. In mode 3, a reception always sets the RI interrupt pending bit, regardless of the state of the ninth bit. If parity is disabled, the SP_STATUS register bit 7 (RB8) contains the ninth data bit. If parity is enabled, then bit 7 (RB8) is the received parity error (RPE) flag.

8.3.2.4 Mode 2 and 3 Timings

Operation in modes 2 and 3 is similar to mode 1 operation. The only difference is that the data consists of 9 bits, so 11-bit packages are transmitted and received. During a reception, the RI flag and the RI interrupt pending bit are set just after the end of the stop bit. During a transmission, the TI flag and the TI interrupt pending bit are set at the beginning of the stop bit. The ninth bit can be used for parity or multiprocessor communications.

8.3.2.5 Multiprocessor Communications

Modes 2 and 3 are provided for multiprocessor communications. In mode 2, the serial port sets the RI interrupt pending bit only when the ninth data bit is set. In mode 3, the serial port sets the RI interrupt pending bit regardless of the value of the ninth bit. The ninth bit is always set in address frames and always cleared in data frames.

One way to use these modes for multiprocessor communication is to set the master processor to mode 3 and the slave processors to mode 2. When the master processor wants to transmit a block of data to one of several slaves, it sends out an address frame that identifies the target slave. Because the ninth bit is set, an address frame interrupts all slaves. Each slave examines the address byte to check whether it is being addressed. The addressed slave switches to mode 3 to receive the data frames, while the slaves that are not addressed remain in mode 2 and are not interrupted.

8.4 PROGRAMMING THE SERIAL PORT

To use the SIO port, you must configure the port pins to serve as special-function signals and set up the SIO channel.

8.4.1 Configuring the Serial Port Pins

Before you can use the serial port, you must configure the associated port pins to serve as special-function signals. Table 8-1 on page 8-2 lists the pins associated with the serial port. Table 8-2 lists the port configuration registers, and Chapter 7, "I/O Ports," explains how to configure the pins.

8.4.2 Programming the Control Register

The SP_CON register (Figure 8-6) selects the communication mode and enables or disables the receiver, parity checking, and nine-bit data transmissions. Selecting a new mode resets the serial I/O port and aborts any transmission or reception in progress on the channel.

8.4.3 Programming the Baud Rate and Clock Source

The SP_BAUD register (Figure 8-7 on page 8-11) selects the clock input for the baud-rate generator and defines the baud rate for all serial I/O modes. This register acts as a control register during write operations and as a down-counter monitor during read operations.

WARNING

Writing to the SP_BAUD register during a reception or transmission can corrupt the received or transmitted data. Before writing to SP_BAUD, check the SP_STATUS register to ensure that the reception or transmission is complete.

SP_CON Address: 1FBBH
Reset State: 00H

The serial port control (SP_CON) register selects the communications mode and enables or disables the receiver, parity checking, and nine-bit data transmission. For the 80C196NU, it also enables or disables the divide-by-two prescaler.

	7							0
8XC196NP	—	—	PAR	TB8	REN	PEN	M1	M0
	7							**0**
80C196NU	—	PRS	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
7	—	Reserved; for compatibility with future devices, write zero to this bit.
6†	PRS	Prescale This bit enables the divide-by-two prescaler. 0 =disable the prescaler 1 =enable the prescaler
5	PAR	Parity Selection Bit Selects even or odd parity. 0 = even parity 1 = odd parity
4	TB8	Transmit Ninth Data Bit This is the ninth data bit that will be transmitted in mode 2 or 3. This bit is cleared after each transmission, so it must be set before SBUF_TX is written. When SP_CON.2 is set, this bit takes on the even parity value.
3	REN	Receive Enable Setting this bit enables the receiver function of the RXD pin. When this bit is set, a high-to-low transition on the pin starts a reception in mode 1, 2, or 3. In mode 0, this bit must be clear for transmission to begin and must be set for reception to begin. Clearing this bit stops a reception in progress and inhibits further receptions.
2	PEN	Parity Enable In modes 1 and 3, setting this bit enables the parity function. This bit must be cleared if mode 2 is used. When this bit is set, TB8 takes the parity value on transmissions. With parity enabled, SP_STATUS.7 becomes the receive parity error bit.

† This bit is reserved on the 8XC196NP. For compatibility with future devices, write zero to this bit.

Figure 8-6. Serial Port Control (SP_CON) Register

SP_CON (Continued)

Address: 1FBBH
Reset State: 00H

The serial port control (SP_CON) register selects the communications mode and enables or disables the receiver, parity checking, and nine-bit data transmission. For the 80C196NU, it also enables or disables the divide-by-two prescaler.

	7							0
8XC196NP	—	—	PAR	TB8	REN	PEN	M1	M0
	7							**0**
80C196NU	—	PRS	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
1:0	M1:0	Mode Selection These bits select the communications mode. **M1 M0** 0 0 mode 0 0 1 mode 1 1 0 mode 2 1 1 mode 3

† This bit is reserved on the 8XC196NP. For compatibility with future devices, write zero to this bit.

Figure 8-6. Serial Port Control (SP_CON) Register (Continued)

SP_BAUD

Address: 1FBCH
Reset State: 0000H

The serial port baud rate (SP_BAUD) register selects the serial port baud rate and clock source. The most-significant bit selects the clock source. The lower 15 bits represent BAUD_VALUE, an unsigned integer that determines the baud rate.

The maximum BAUD_VALUE is 32,767 (7FFFH). In asynchronous modes 1, 2, and 3, the minimum BAUD_VALUE is 0000H when using the internal clock source (f) and 0001H when using T1CLK. In synchronous mode 0, the minimum BAUD_VALUE is 0001H for transmissions and 0002H for receptions.

15							**8**
CLKSRC	BV14	BV13	BV12	BV11	BV10	BV9	BV8
7							**0**
BV7	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
15	CLKSRC	Serial Port Clock Source This bit determines whether the serial port is clocked from an internal or an external source. 0 = signal on the T1CLK pin (external source) 1 = internal operating frequency (f)
14:0	BV14:0	Baud Rate These bits constitute the BAUD_VALUE. Use the following equations to determine the BAUD_VALUE for a given baud rate. Synchronous mode 0:† $BAUD\_VALUE = \frac{f}{Baud\ Rate \times 2} - 1$ or $\frac{T1CLK}{Baud\ Rate}$ Asynchronous modes 1, 2, and 3: $BAUD\_VALUE = \frac{f}{Baud\ Rate \times 16} - 1$ or $\frac{T1CLK}{Baud\ Rate \times 8}$ † For mode 0 receptions, the BAUD_VALUE must be 0002H or greater. Otherwise, the resulting data in the receive shift register will be incorrect.

Figure 8-7. Serial Port Baud Rate (SP_BAUD) Register

CAUTION

For mode 0 receptions, the BAUD_VALUE must be 0002H or greater. Otherwise, the resulting data in the receive shift register will be incorrect.

The reason for this restriction is that the receive shift register is clocked from an internal signal rather than the signal on TXD. Although these two signals are normally synchronized, the internal signal generates one clock before the first pulse transmitted by TXD and this first clock signal is not synchronized with TXD. This clock signal causes the receive shift register to shift in whatever data is present on the RXD pin. This data is treated as the least-significant bit (LSB) of the reception. The reception then continues in the normal synchronous manner, but the data received is shifted left by one bit because of the false LSB. The seventh data bit transmitted is received as the most-significant bit (MSB), and the transmitted MSB is never shifted into the receive shift register.

Using the internal peripheral clock at 25 MHz, the maximum baud rate is 4.17 Mbaud for mode 0 receptions and 6.25 Mbaud for mode 0 transmissions. The maximum baud rate for modes 1, 2, and 3 is 1.56 Mbaud for both receptions and transmissions. For the 80C196NU using the internal peripheral clock at 50 MHz, the maximum baud rates are doubled: 12.5 Mbaud for mode 0 transmissions, 8.33 Mbaud for mode 0 receptions, and 3.13 Mbaud for modes 1, 2, and 3.

Table 8-3 shows the SP_BAUD values for common baud rates when using a 25 MHz internal clock. These values also apply to the 80C196NU at 50 MHz with the prescaler enabled. Table 8-3 shows the SP_BAUD value for 9600 baud when using a 50 MHz clock input with the prescaler disabled. Because of rounding, the BAUD_VALUE formula is not exact and the resulting baud rate is slightly different than desired. The tables show the percentage of error when using the sample SP_BAUD values. In most cases, a serial link will work with up to 5.0% difference in the receiving and transmitting baud rates.

Table 8-3. SP_BAUD Values When Using the Internal Clock at 25 MHz

Baud Rate	SP_BAUD Register Value (Note 1)		% Error	
	Mode 0	Mode 1, 2, 3	Mode 0	Mode 1, 2, 3
9600	8515H	80A2H	0	0.15
4800	8A2BH	8144H	0	0.16
2400	9457H	828AH	0	0
1200	A8AFH	8515H	0	0
300	(Note 2)	9457H	(Note 2)	0

NOTES:
1. Bit 15 is always set when the internal peripheral clock is selected as the clock source for the baud-rate generator.
2. For mode 0 operation at 25 MHz, the minimum baud rate is 381.47 (BAUD_VALUE = 7FFFH). For mode 0 operation at 300 baud, the maximum internal clock frequency is 19.6608 MHz (BAUD_VALUE = 7FFFH).

Table 8-4. SP_BAUD Values When Using the Internal Clock at 50 MHz (80C196NU Only)

Baud Rate	SP_BAUD Register Value†		% Error	
	Mode 0	Mode 1, 2, 3	Mode 0	Mode 1, 2, 3
9600	8A2CH	8145H	0	0.15

†Bit 15 is always set when the internal peripheral clock is selected as the clock source for the baud-rate generator.

8.4.4 Enabling the Serial Port Interrupts

The serial port has both a transmit interrupt (TI) and a receive interrupt (RI). To enable an interrupt, set the corresponding mask bit in the interrupt mask register (see Table 8-2 on page 8-2) and execute the EI instruction to globally enable servicing of interrupts. See Chapter 6, “Standard and PTS Interrupts,” for more information about interrupts.

8.4.5 Determining Serial Port Status

You can read the SP_STATUS register (Figure 8-8) to determine the status of the serial port. Reading SP_STATUS **clears all bits** except TXE. For this reason, we recommend that you copy the contents of the SP_STATUS register into a shadow register and then execute bit-test instructions such as JBC and JBS on the shadow register. Otherwise, executing a bit-test instruction clears the flags, so any subsequent bit-test instructions will return false values. You can also read the interrupt pending register (see Table 8-2 on page 8-2) to determine the status of the serial port interrupts.

SP_STATUS Address: 1FB9H
Reset State: 0BH

The serial port status (SP_STATUS) register contains bits that indicate the status of the serial port.

7							0
RPE/RB8	RI	TI	FE	TXE	OE	—	—

Bit Number	Bit Mnemonic	Function
7	RPE/RB8	Received Parity Error/Received Bit 8 RPE is set if parity is disabled (SP_CON.2 = 0) and the ninth data bit received is high. RB8 is set if parity is enabled (SP_CON.2 = 1) and a parity error occurred. Reading SP_STATUS clears this bit.
6	RI	Receive Interrupt This bit is set when the last data bit is sampled. Reading SP_STATUS clears this bit. This bit need **not** be clear for the serial port to receive data.
5	TI	Transmit Interrupt This bit is set at the beginning of the stop bit transmission. Reading SP_STATUS clears this bit.
4	FE	Framing Error This bit is set if a stop bit is not found within the appropriate period of time. Reading SP_STATUS clears this bit.
3	TXE	SBUF_TX Empty This bit is set if the transmit buffer is empty and ready to accept up to two bytes. It is cleared when a byte is written to SBUF_TX.
2	OE	Overrun Error This bit is set if data in the receive shift register is loaded into SBUF_RX before the previous bit is read. Reading SP_STATUS clears this bit.
1:0	—	Reserved. These bits are undefined.

Figure 8-8. Serial Port Status (SP_STATUS) Register

The receiver checks for a valid stop bit. Unless a stop bit is found within the appropriate time, the framing error (FE) bit in the SP_STATUS register is set. When the stop bit is detected, the data in the receive shift register is loaded into SBUF_RX and the receive interrupt (RI) flag is set. If this happens before the previous byte in SBUF_RX is read, the overrun error (OE) bit is set. SBUF_RX always contains the latest byte received; it is never a combination of the last two bytes.

The receive interrupt (RI) flag indicates whether an incoming data byte has been received. The transmit interrupt (TI) flag indicates whether a data byte has finished transmitting. These flags also set the corresponding bits in the interrupt pending register. A reception or transmission sets the RI or TI flag in SP_STATUS and the corresponding interrupt pending bit. However, a software write to the RI or TI flag in SP_STATUS has no effect on the interrupt pending bits and does not cause an interrupt. Similarly, reading SP_STATUS clears the RI and TI flags, but does not clear the corresponding interrupt pending bits. The RI and TI flags in the SP_STATUS and the corresponding interrupt pending bits can be set even if the RI and TI interrupts are masked.

The transmitter empty (TXE) bit is set if SBUF_TX and its buffer are empty and ready to accept up to two bytes. TXE is cleared as soon as a byte is written to SBUF_TX. One byte may be written if TI alone is set. By definition, if TXE has just been set, a transmission has completed and TI is set.

The received parity error (RPE) flag or the received bit 8 (RB8) flag applies for parity enabled or disabled, respectively. If parity is enabled, RPE is set if a parity error is detected. If parity is disabled, RB8 is the ninth data bit received in modes 2 and 3.

9

Pulse-width Modulator

CHAPTER 9
PULSE-WIDTH MODULATOR

The pulse-width modulator (PWM) module has three output pins, each of which can output a PWM signal with a fixed frequency and a variable duty cycle. These outputs can be used to drive motors that require an unfiltered PWM waveform for optimal efficiency, or they can be filtered to produce a smooth analog signal.

This chapter provides a functional overview of the pulse-width modulator module, describes how to program it, and provides sample circuitry for converting the PWM outputs to analog signals. For detailed descriptions of the signals and registers discussed in this chapter, please refer to Appendix B, “Signal Descriptions” and Appendix C, “Registers.”

9.1 PWM FUNCTIONAL OVERVIEW

The PWM module has three channels, each of which consists of a control register (PWM*x*_CONTROL, where *x* is 0, 1, or 2), a buffer, a comparator, an RS flip-flop, and an output pin. Two other components, an eight-bit counter and a clock prescaler, are shared across the PWM module’s three channels, completing the circuitry (see Figures 9-1 and 9-2).

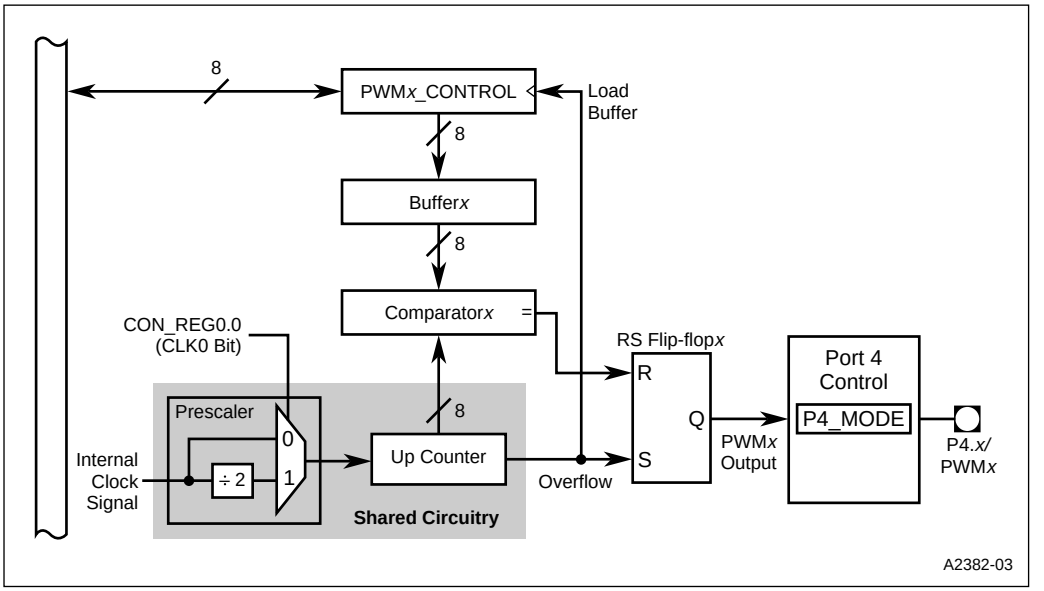


Figure 9-1. PWM Block Diagram (8XC196NP Only)

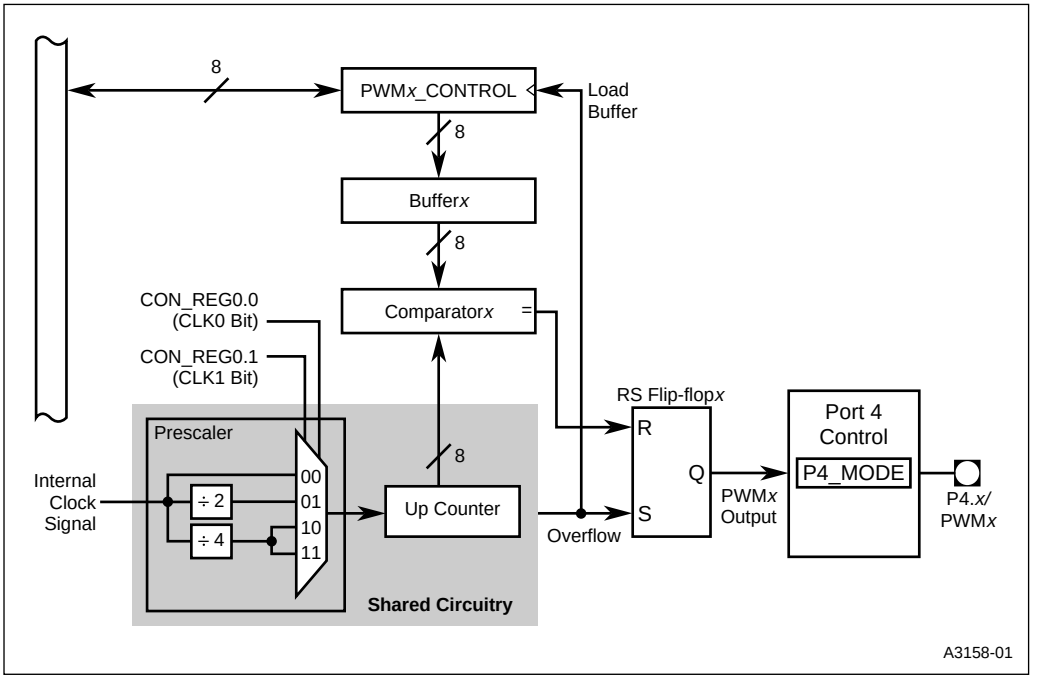


Figure 9-2. PWM Block Diagram (80C196NU Only)

9.2 PWM SIGNALS AND REGISTERS

Table 9-1 describes the PWM's signals and Table 9-2 briefly describes the control and status registers.

Table 9-1. PWM Signals

Port Pin	PWM Signal	PWM Signal Type	Description
P4.0	PWM0	O	Pulse-width modulator 0 output with high-drive capability.
P4.1	PWM1	O	Pulse-width modulator 1 output with high-drive capability.
P4.2	PWM2	O	Pulse-width modulator 2 output with high-drive capability.

Table 9-2. PWM Control and Status Registers

Mnemonic	Address	Description
CON_REG0	1FB6H	PWM Control Register This register controls the clock prescaler. Bit 0 (CLK0) controls the output period of the PWM channels by enabling or disabling the divide-by-two clock prescaler (8XC196NP only). Bits 0 and 1 (CLK0, CLK1) control the output period of the PWM channels by enabling or disabling the divide-by-two or divide-by-four clock prescaler (80C196NU only).
PWM0_CONTROL PWM1_CONTROL PWM2_CONTROL	1FB0H 1FB2H 1FB4H	PWM Duty Cycle This register controls the PWM duty cycle. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).
P4_DIR	1FDBH	Port 4 Direction The P4_DIR register determines the I/O mode for each port 4 pin. The register settings for an open-drain output or a high-impedance input are identical. An open-drain output configuration requires an external pull-up. A high-impedance input configuration requires that the corresponding bit in P4_REG be set. This port has a higher drive capability than the other ports in order to support PWM high-drive output requirements.
P4_MODE	1FD9H	Port 4 Mode Each bit in this register determines whether the corresponding pin functions as a standard I/O port pin or is used for a special-function signal.
P4_PIN	1FDFH	Port 4 Pin State P4_PIN contains the current state of each port pin, regardless of the pin mode setting.
P4_REG	1FDDH	Port 4 Output Data P4_REG contains data to be driven out by the respective pins. When a port pin is configured as an input, the corresponding bit in P4_REG must be set.

9.3 PWM OPERATION

For the 8XC196NP, CON_REG0.0 (CLK0) controls the PWM output frequency by enabling or disabling the divide-by-two clock prescaler. Enabling the prescaler causes the 8-bit counter to increment once every two state times; disabling it causes the counter to increment once every state time.

For the 80C196NU, two bits control the PWM output frequency, CON_REG0.0 (CLK0) and CON_REG0.1 (CLK1). The two bits control the PWM output frequency by enabling or disabling the divide-by-two or divide-by-four clock prescaler.

Each control register (PWM*x*_CONTROL; *x* = 0, 1, or 2) controls the duty cycle (the pulsewidth stated as a percentage of the period) of the corresponding PWM output. Each control register contains an 8-bit value that is loaded into a buffer when the 8-bit counter rolls over from FFH to 00H. The comparators compare the contents of the buffers to the counter value. Since the value written to the control register is buffered, you can write a new 8-bit value to PWM*x*_CONTROL at any time. However, the comparators do not recognize the new value until the counter has expired the remainder of the current 8-bit count. The new value is used during the next PWM output period.

The counter continually increments until it rolls over to 00H, at which time the PWM output is driven high and the contents of the control registers are loaded into the buffers. The PWM output remains high until the counter value matches the value in the buffer, at which time the output is pulled low. When the counter resets again (i.e., when an overflow occurs) the output is switched high. (Loading PWM*x*_CONTROL with 00H forces the output to remain low.) Figure 9-3 shows typical PWM output waveforms.

The PWM can generate a duty cycle ranging in length from 0% to 99.6% of the pulse. To determine the desired duty cycle measurement, you must apply a multiplier (2, 4, or 8) to the PWM*x*_CONTROL value to compensate for the divided input frequency from the divide-by-two circuitry. (See Chapter 2, "Architectural Overview," for additional information.)

Clearing CON_REG0.0 (CLK0) disables the prescaler, generating a pulse that is 512 state times in length. With the prescaler disabled, the correct multiplier is 2.

Setting CON_REG0.0 (CLK0) enables the PWM's divide-by-two clock prescaler, generating a pulse that is 1,024 state times in length. With the divide-by-two clock prescaler enabled, the correct multiplier is 4. For example, assume that CLK0 is set and the value you write to the PWM*x*_CONTROL register is 19H (25 decimal). To arrive at the appropriate duty cycle, you must multiply the value stored in PWM*x*_CONTROL by 4, then divide that result by the total pulse length (1,024). This calculation results in a duty cycle value of approximately 10% (.0977).

For the 80C196NU, setting CON_REG0.1 (CLK1) enables the divide-by-four clock prescaler, generating a pulse that is 2,048 state times in length. With the divide-by-four prescaler enabled, the correct multiplier is 8. (When CON_REG0.1 is set, the divide-by-four clock prescaler is enabled and CON_REG0.0 is ignored.)

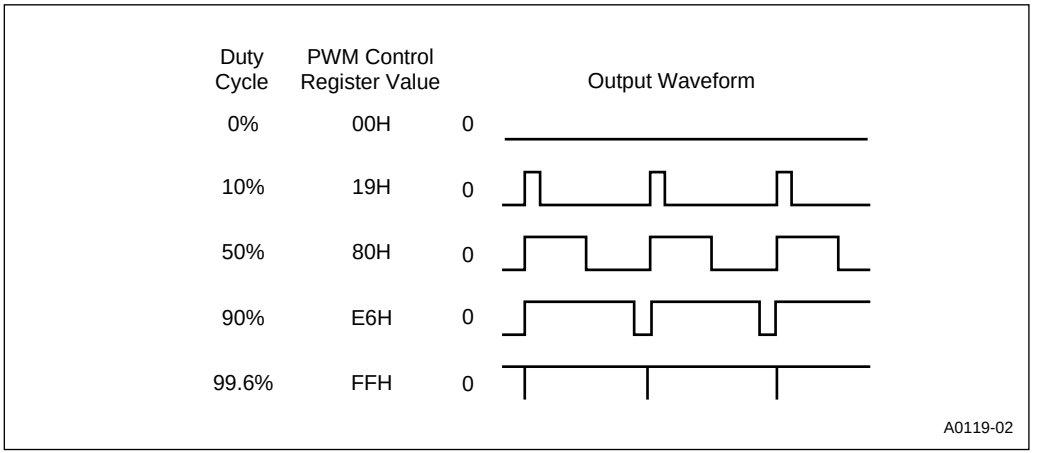


Figure 9-3. PWM Output Waveforms

9.4 PROGRAMMING THE FREQUENCY AND PERIOD

The PWM module provides two selectable, fixed PWM output frequencies for a specified internal operating frequency (f). Table 9-3 shows the PWM output frequencies for common operating frequencies on the 8XC196NP. The value of CON_REG0.0 determines the output frequency by enabling or disabling the clock prescaler. Use the following formulas to calculate the output frequency (F_{PWM}) or output period (T_{PWM}).

	Clock Prescaler Disabled	**÷2 Clock Prescaler Enabled**	**÷4 Clock Prescaler† Enabled**
F_{PWM} (in MHz) =	$\frac{f}{512}$	$\frac{f}{1024}$	$\frac{f}{2048}$
T_{PWM} (in µs) =	$\frac{512}{f}$	$\frac{1024}{f}$	$\frac{2048}{f}$

† 80C196NU only.

For the 80C196NU, the PWM module provides three selectable, fixed PWM output frequencies for a specified internal operating frequency (f). Table 9-3 shows the PWM output frequencies for common operating frequencies. The value of bits 0 and 1 in the CON_REG0 register determines the output frequency by enabling or disabling the divide-by-two or divide-by-four clock prescaler.

NOTE

Use the EPA module to produce variable PWM output frequencies (see "Operating in Compare Mode" on page 10-12).

Table 9-3. PWM Output Frequencies (8XC196NP)

CLK0	f		
	16 MHz	20 MHz	25 MHz
0	31.25 kHz	39.06 kHz	48.83 kHz
1	15.63 kHz	19.53 kHz	24.41 kHz

Table 9-4. PWM Output Frequencies (80C196NU)

CLK1	CLK0	f		
		12.5 MHz	25 MHz	50 MHz
0	0	24.41 kHz	48.83 kHz	97.66 kHz
0	1	12.21 kHz	24.41 kHz	48.83 kHz
1	X	6.10 kHz	12.21 kHz	24.41 kHz

CON_REG0 Address: 1FB6H
Reset State: FEH

The control (CON_REG0) register controls the clock prescaler for the three pulse-width modulators (PWM0–PWM2).

	7							0
8XC196NP	—	—	—	—	—	—	—	CLK0†
	7							**0**
80C196NU	—	—	—	—	—	—	CLK1	CLK0

Bit Number	Bit Mnemonic	Function
7:1 (NP) 7:2 (NU)	—	Reserved; for compatibility with future devices, write zeros to these bits.
0 (NP)	CLK0	Enable PWM Clock Prescaler This bit controls the PWM output period by enabling or disabling the clock prescaler (divide-by-two) on the three pulse width modulators (PWM2:0). 0 = disable; PWM output period is 512 state times 1 = enable; PWM output period is 1024 state times
1:0 (NU)	CLK1:0	Enable PWM Clock Prescaler These bits control the PWM output period on the three pulse-width modulators (PWM2:0). **CLK1 CLK0** 0 0 disable clock prescaler 0 1 enable divide-by-two prescaler; PWM output period is 1024 state times 1 X enable divide-by-four prescaler; PWM output period is 2048 state times

† This bit was called SLOW_PWM in earlier documentation for the 8XC196NP.

Figure 9-4. Control (CON_REG0) Register

9.5 PROGRAMMING THE DUTY CYCLE

The value written to the PWM*x*_CONTROL register controls the width of the high pulse, effectively controlling the duty cycle. The 8-bit value written to the control register is loaded into a buffer, and this value is used during the next period. Use the following formula to calculate a desired pulsewidth by extrapolating an appropriate value for PWM*x*_CONTROL from the range 00–FFH, and then write the value to the PWM*x*_CONTROL register.

		Clock Prescaler Disabled	÷2 Clock Prescaler Enabled	÷4 Clock Prescaler† Enabled
Pulsewidth (in µs)	=	$\frac{PWMx\_CON \times 2}{f}$	$\frac{PWMx\_CON \times 4}{f}$	$\frac{PWMx\_CON \times 8}{f}$
Duty Cycle (in %)	=	$\frac{Pulsewidth}{T_{PWM}} \times 100$		

where:

PWM*x*_CON	=	8-bit value to load into the PWM*x*_CONTROL register
Pulsewidth	=	width of each high pulse
f	=	operating frequency, in MHz
T_{PWM}	=	output period on the PWM pin, in µs

† 80C196NU only.

PWM*x*_CONTROL
***x* = 0–2**

Address: Table 9-2
Reset State: 00H

The PWM control (PWM*x*_CONTROL) register determines the duty cycle of the PWM *x* channel. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).

7							0
PWM Duty Cycle							

Bit Number	Function
7:0	PWM Duty Cycle This register controls the PWM duty cycle. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).

Figure 9-5. PWM Control (PWM*x*_CONTROL) Register

9.5.1 Sample Calculations

For example, assume that the operating frequency equals 25 MHz, the desired period of the PWM output waveform is either 20.48 µs (512 state times) if the divide-by-two prescaler is disabled or 40.96 µs (1,024 state times) if the prescaler is enabled. If PWM*x*_CONTROL equals 8AH (138 decimal), the pulsewidth is held high for 11.04 µs (and low for 9.44 µs) of the total 20.48 µs period, resulting in a duty cycle of approximately 54%. If the prescaler is enabled, the same values would produce a period of 40.96 µs with the pulsewidth being held high for 22.08 µs (and low for 18.88 µs), for the same duty cycle, approximately 54%.

9.5.2 Enabling the PWM Outputs

Each PWM output is multiplexed with a port pin, so you must configure it as a special-function output signal before using the PWM function. To do so, follow this sequence:

1. Clear the corresponding bit of P4_DIR (see Table 9-5).
2. Set the corresponding bit of P4_MODE (see Table 9-5).
3. Set or clear the corresponding bit of P4_REG (see Table 9-5).

Table 9-5 shows the alternate port function along with the register setting that selects the PWM output instead of the port function.

Table 9-5. PWM Output Alternate Functions

PWM Output	Alternate Port Function	PWM Output Enabled When:
PWM0	P4.0	P4_DIR.0 = 0, P4_MODE.0 = 1, P4_REG = X
PWM1	P4.1	P4_DIR.1 = 0, P4_MODE.1 = 1, P4_REG = X
PWM2	P4.2	P4_DIR.2 = 0, P4_MODE.2 = 1, P4_REG = X

9.5.3 Generating Analog Outputs

The PWM modules can generate a rectangular pulse train that varies in duty cycle and period. Filtering this output will create a smooth analog signal. To make a signal swing over the desired analog range, first buffer the signal and then filter it with either a simple RC network or an active filter. Figure 9-6 is a block diagram of the type of circuit needed to create the smooth analog signal.

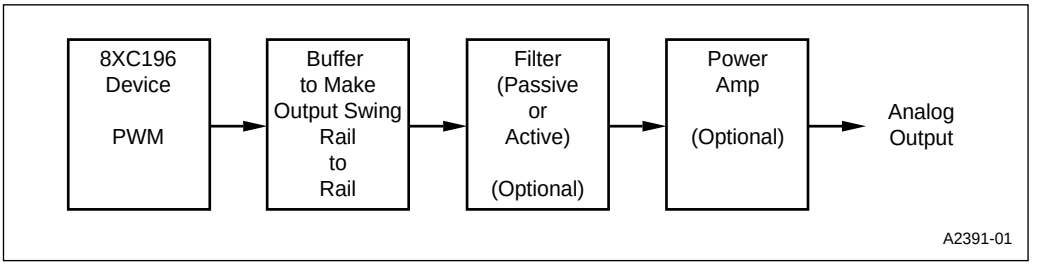


Figure 9-6. D/A Buffer Block Diagram

Figure 9-7 shows a sample circuit used for low output currents (less than 100 µA). Consider temperature and power-supply drift when selecting components for the external D/A circuitry. With proper components, a highly accurate 8-bit D/A converter can be made using the PWM.

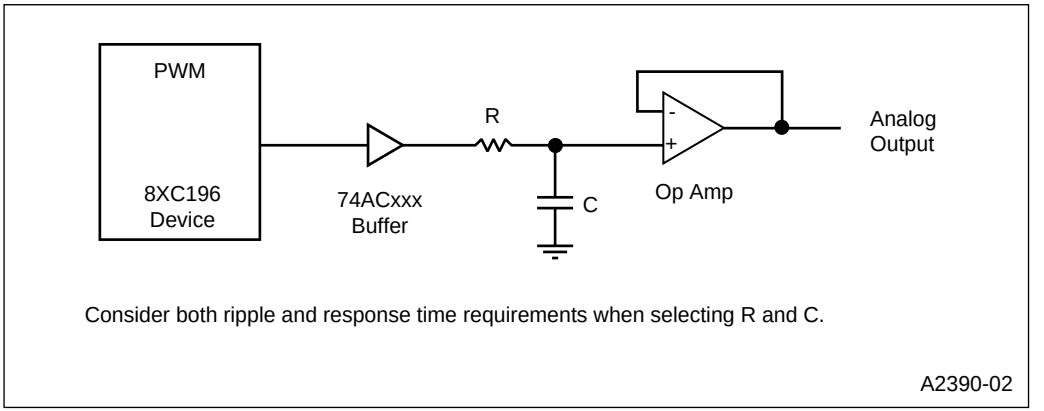


Figure 9-7. PWM to Analog Conversion Circuitry

10

Event Processor Array (EPA)

CHAPTER 10
EVENT PROCESSOR ARRAY (EPA)

Control applications often require high-speed event control. For example, the controller may need to periodically generate pulse-width modulated outputs or an interrupt. In another application, the controller may monitor an input signal to determine the status of an external device. The event processor array (EPA) was designed to reduce the CPU overhead associated with these types of event control. This chapter describes the EPA and its timers and explains how to configure and program them.

10.1 EPA FUNCTIONAL OVERVIEW

The EPA performs input and output functions associated with two timer/counters, timer 1 and timer 2 (Figure 10-1). In the input mode, the EPA monitors an input pin for an event: a rising edge, a falling edge, or an edge in either direction. When the event occurs, the EPA records the value of the timer/counter, so that the event is tagged with a time. This is called an *input capture*. Input captures are buffered to allow two captures before an overrun occurs. In the output mode, the EPA monitors a timer/counter and compares its value with a value stored in a register. When the timer/counter value matches the stored value, the EPA can trigger an event: a timer reset or an output event (set a pin, clear a pin, toggle a pin, or take no action). This is called an *output compare*. Each input capture or an output compare sets an interrupt pending bit. This bit can optionally cause an interrupt. The EPA has four capture/compare channels, EPA3:0.

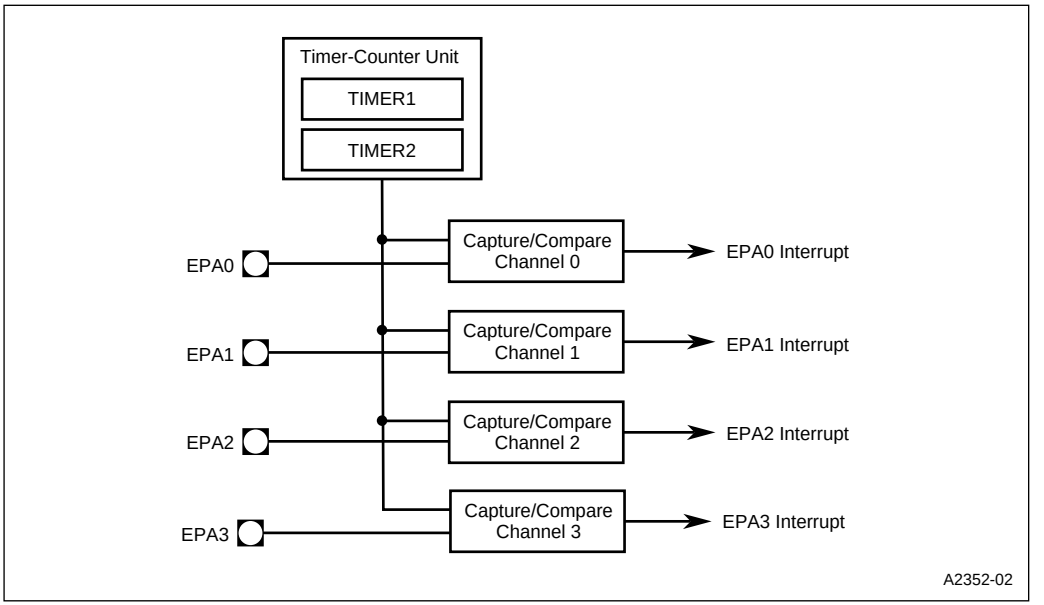


Figure 10-1. EPA Block Diagram

10.2 EPA AND TIMER/COUNTER SIGNALS AND REGISTERS

Table 10-1 describes the EPA and timer/counter input and output signals. Each signal is multiplexed with a port pin as shown in the first column. Table 10-2 briefly describes the registers for the EPA capture/compare channels and timer/counters.

Table 10-1. EPA and Timer/Counter Signals

Port Pin	EPA Signal(s)	EPA Signal Type	Description
P1.3:0	EPA3:0	I/O	High-speed input/output for capture/compare channels 0–3.
P1.4	T1CLK	I	External clock source for timer 1.
P1.5	T1DIR	I	External direction control for timer 1.
P1.6	T2CLK	I	External clock source for timer 2.
P1.7	T2DIR	I	External direction control for timer 2.

Table 10-2. EPA Control and Status Registers

Mnemonic	Address	Description
EPA_MASK	1F9CH	EPA Mask Four bits (OVR0, OVR1, OVR2, and OVR3) in this 8-bit register enable and disable (mask) the individual capture overrun interrupt sources associated with capture/compare channels EPA3:0.
EPA_PEND	1F9EH	EPA Pending Four bits (OVR0, OVR1, OVR2, and OVR3) in this 8-bit register indicate an overrun status for the associated capture/compare channels, EPA3:0. OVR0 and OVR1 are multiplexed to share one interrupt pending bit (OVR0_1) in INT_PEND1; OVR2 and OVR3 are multiplexed to share another interrupt pending bit (OVR2_3) in INT_PEND1.
EPA0_CON EPA1_CON EPA2_CON EPA3_CON	1F80H 1F84H 1F88H 1F8CH	EPA*x* Capture/Compare Control These registers control the functions of the capture/compare channels. EPA1_CON and EPA3_CON require an extra byte because they contain an additional bit for PWM remap mode. These two registers must be addressed as words; the others can be addressed as bytes.
EPA0_TIME EPA1_TIME EPA2_TIME EPA3_TIME	1F82H 1F86H 1F8AH 1F8EH	EPA*x* Capture/Compare Time In capture mode, these registers contain the captured timer value. In compare mode, these registers contain the time at which an event is to occur. In capture mode, these registers are buffered to allow two captures before an overrun occurs. However, they are not buffered in compare mode.
INT_MASK	0008H	Interrupt Mask Three bits in this 8-bit register (OVRTM1, OVRTM2, and EPA0) enable and disable (mask) the three interrupts associated with the corresponding bits in INT_PEND register.
INT_MASK1	0013H	Interrupt Mask 1 Five bits in this 8-bit register (EPA1, EPA2, EPA3, OVR0_1, and OVR2_3) enable and disable (mask) the five interrupts associated with the corresponding bits in INT_PEND1 register.
INT_PEND	0009H	Interrupt Pending Any set bit in this 8-bit register indicates a pending interrupt. The three bits associated with EPA interrupts are OVRTM1, OVRTM2, and EPA0.
INT_PEND1	0012H	Interrupt Pending 1 Any set bit in this 8-bit register indicates a pending interrupt. The five bits associated with EPA interrupts are EPA1, EPA2, EPA3, OVR0_1, and OVR2_3.
P1_DIR	1FD2H	Port 1 Direction Each bit of P1_DIR controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as an input or open-drain output. (Open-drain outputs require external pull-ups.)

Table 10-2. EPA Control and Status Registers (Continued)

Mnemonic	Address	Description
P1_MODE	1FD0H	Port 1 Mode Each bit of P1_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a standard I/O port pin.
P1_PIN	1FD6H	Port 1 Input Each bit of P1_PIN reflects the current state of the corresponding pin, regardless of the pin configuration.
P1_REG	1FD4H	Port 1 Data Output For an input, set the corresponding P1_REG bit. For an output, write the data to be driven out by each pin to the corresponding bit of P1_REG. When a pin is configured as standard I/O (P1_MODE.*y* = 0), the result of a CPU write to P*x*_REG is immediately visible on the pin. When a pin is configured as a special-function signal (P1_MODE.*y* = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to P1_REG, but the pin is unaffected until it is switched back to its standard I/O function. This feature allows software to configure a pin as standard I/O (clear P1_MODE.*y*), initialize or overwrite the pin value, then configure the pin as a special-function signal (set P1_MODE.*y*). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.
T1CONTROL	1F90H	Timer 1 Control This register enables/disables timer 1, controls whether it counts up or down, selects the clock source and direction, and determines the clock prescaler setting.
T2CONTROL	1F94H	Timer 2 Control This register enables/disables timer 2, controls whether it counts up or down, selects the clock source and direction, and determines the clock prescaler setting.
TIMER1	1F92H	Timer 1 Value This register contains the current value of timer 1.
TIMER2	1F96H	Timer 2 Value This register contains the current value of timer 2.

10.3 TIMER/COUNTER FUNCTIONAL OVERVIEW

The EPA has two 16-bit up/down timer/counters, timer 1 and timer 2, which can be clocked internally or externally. Each is called a *timer* if it is clocked internally and a *counter* if it is clocked externally. Figure 10-2 illustrates the timer/counter structure.

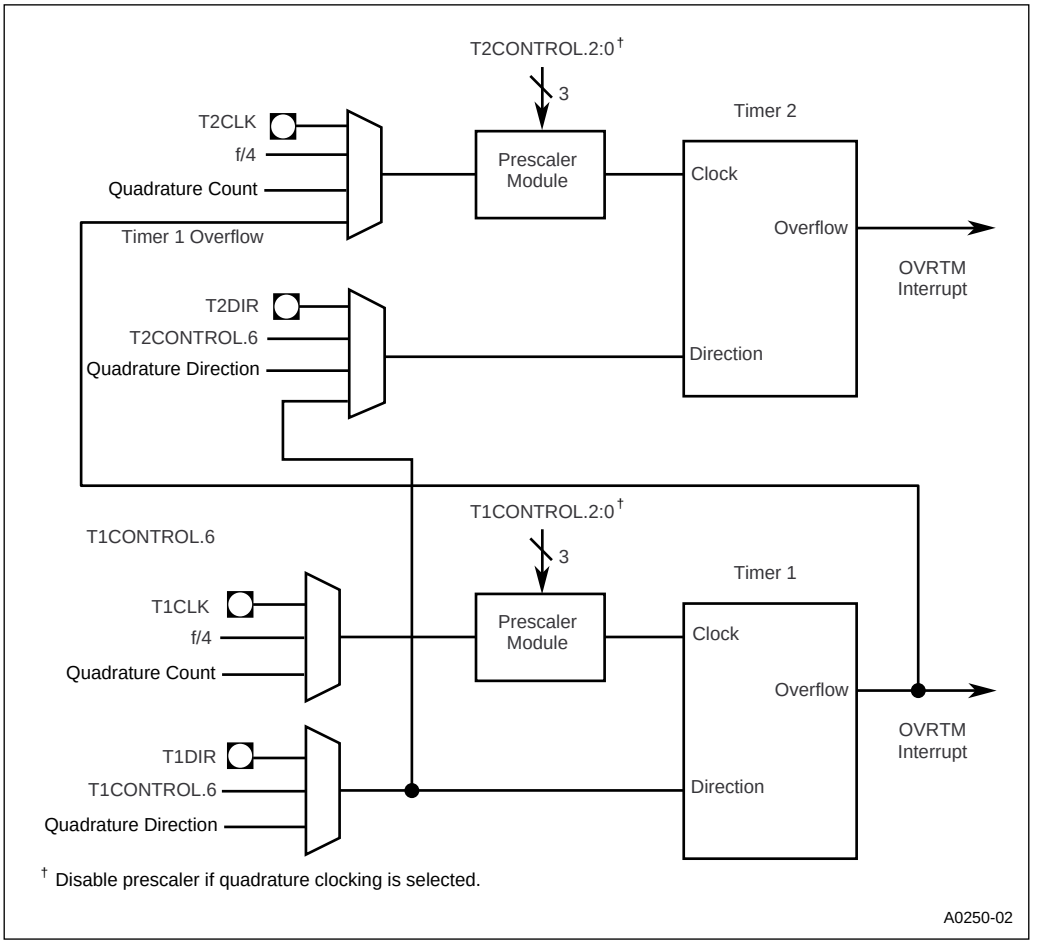


Figure 10-2. EPA Timer/Counters

The timer/counters can be used as time bases for input captures, output compares, and programmed interrupts (software timers). When a counter increments from FFFEH to FFFFH or decrements from 0001H to 0000H, the counter-overflow interrupt pending bit is set. This bit can optionally cause an interrupt. The clock source, direction-control source, count direction, and resolution of the input capture or output compare are all programmable (see "Programming the Timers" on page 10-15). The maximum count rate is one-half the internal clock rate, or f/4 (see "Internal Timing" on page 2-7). This provides a minimum resolution for an input capture or output compare of 160 ns (at f = 25 MHz) for 8XC196NP and 80 ns (at f = 50 MHz) for the 80C196NU.

$$\text{resolution} = \frac{4 \times \text{prescaler\_divisor}}{f}$$

where:

prescaler_divisor — is the clock prescaler divisor from the T*x*CONTROL registers (see "Timer 1 Control (T1CONTROL) Register" on page 10-16 and "Timer 2 Control (T2CONTROL) Register" on page 10-17).

f — is the internal operating frequency. See "Internal Timing" on page 2-7 for details.

10.3.1 Cascade Mode (Timer 2 Only)

Timer 2 can be used in cascade mode. In this mode, the timer 1 overflow output is used as the timer 2 clock input. Either the direction control bit of the timer 2 control register or the direction control assigned to timer 1 controls the count direction. This method, called *cascading*, can provide a slow clock for idle mode timeout control or for slow pulse-width modulation (PWM) applications (see "Generating a Low-speed PWM Output" on page 10-12).

10.3.2 Quadrature Clocking Mode

Both timer 1 and timer 2 can be used in quadrature clocking mode. This mode uses the T*x*CLK and T*x*DIR pins as quadrature inputs, as shown in Figure 10-3. External quadrature-encoded signals (two signals at the same frequency that differ in phase by 90°) are input, and the timer increments or decrements by one count on each rising edge and each falling edge. Because the T*x*CLK and T*x*DIR inputs are sampled by the internal phase clocks, transitions must be separated by at least two state times for proper operation. The count is clocked by PH2, which is PH1 delayed by one-half period. The sequence of the signal edges and levels controls the count direction. Refer to Figure 10-4 and Table 10-3 for sequencing information.

A typical source of quadrature-encoded signals is a shaft-angle decoder, shown in Figure 10-3. Its output signals X and Y are input to T*x*CLK and T*x*DIR, which in turn output signals X_internal and Y_internal. These signals are used in Figure 10-4 and Table 10-3 to describe the direction of the shaft.

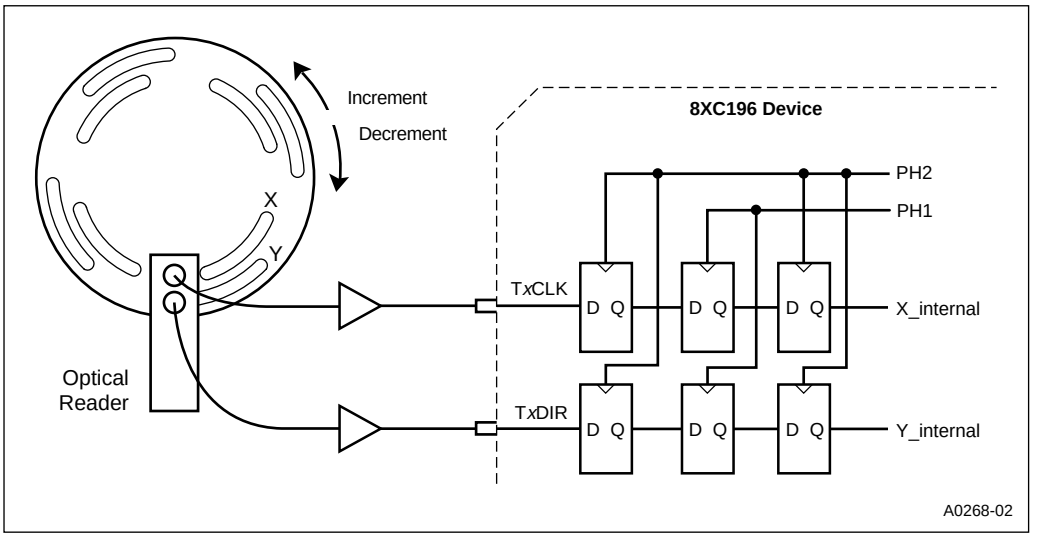


Figure 10-3. Quadrature Mode Interface

Table 10-3. Quadrature Mode Truth Table

State of X_internal (T*x*CLK)	State of Y_internal (T*x*DIR)	Count Direction
↑	0	Increment
↓	1	Increment
0	↓	Increment
1	↑	Increment
↓	0	Decrement
↑	1	Decrement
0	↑	Decrement
1	↓	Decrement

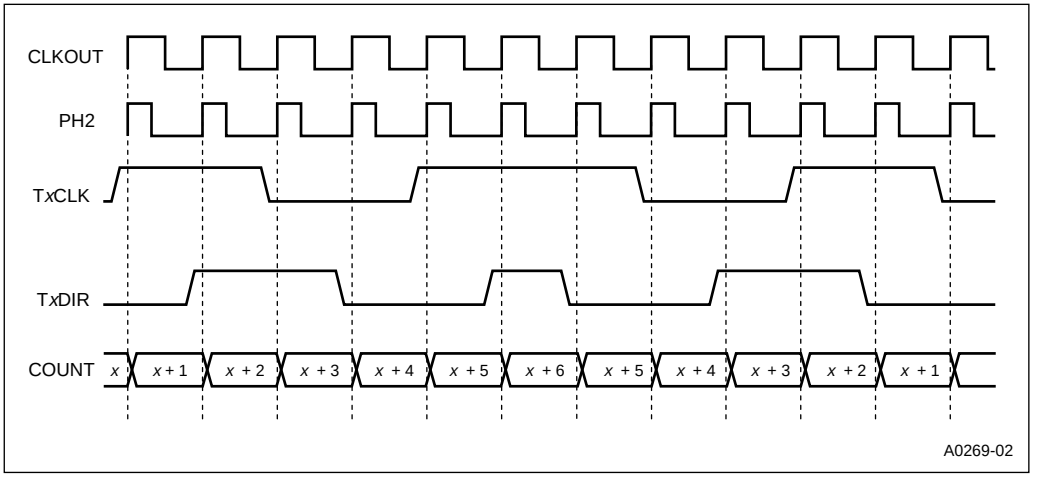


Figure 10-4. Quadrature Mode Timing and Count

10.4 EPA CHANNEL FUNCTIONAL OVERVIEW

The EPA has four programmable capture/compare channels that can perform the following tasks.

- capture the current timer value when a specified transition occurs on the EPA pin
- clear, set, or toggle the EPA pin when the timer value matches the programmed value in the event-time register
- generate an interrupt when a capture or compare event occurs
- generate an interrupt when a capture overrun occurs
- reset its own base timer in compare mode
- reset the opposite timer in both compare and capture mode

Each EPA channel has a control register, EPA*x*_CON (capture/compare channel); an event-time register, EPA*x*_TIME (capture/compare channel); and a timer input (Figure 10-5). The control register selects the timer, the mode, and either the event to be captured or the event that is to occur. The event-time register holds the captured timer value in capture mode and the event time in compare mode. See "Programming the Capture/Compare Channels" on page 10-18 for configuration information.

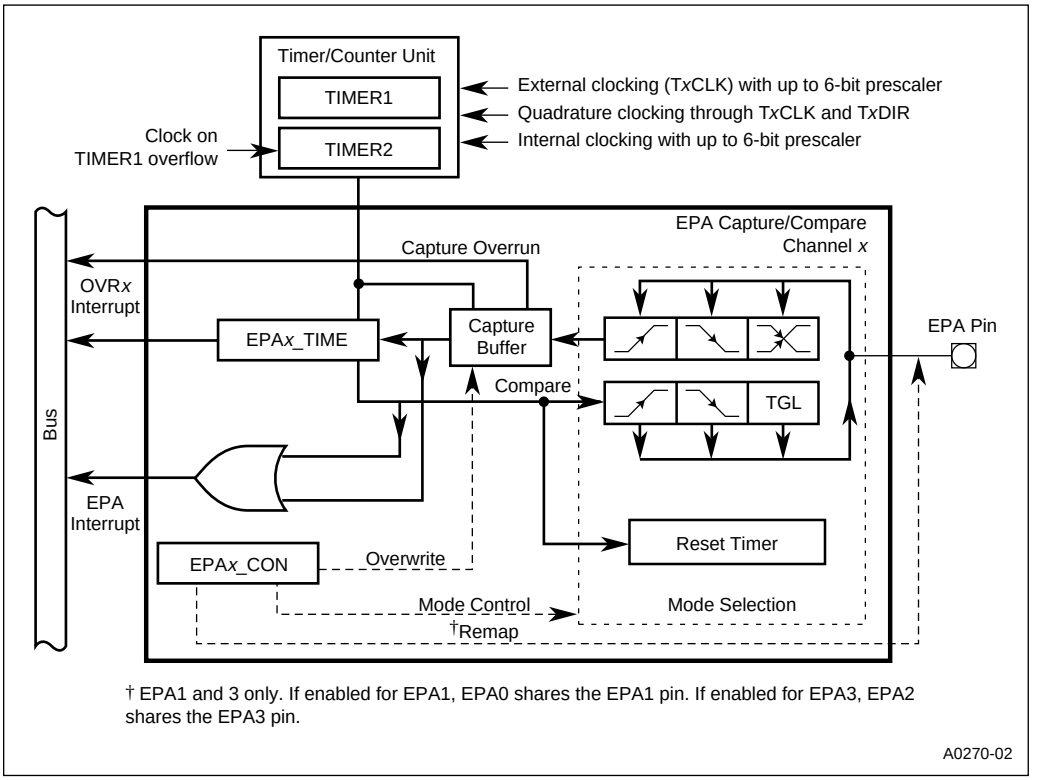


Figure 10-5. A Single EPA Capture/Compare Channel

10.4.1 Operating in Capture Mode

In capture mode, when a valid event occurs on the pin, the value of the selected timer is captured into a buffer. The timer value is then transferred from the buffer to the EPA*x*_TIME register, which sets the EPA interrupt pending bit as shown in Figure 10-6. If enabled, an interrupt is generated. If a second event occurs before the CPU reads the first timer value in EPA*x*_TIME, the current timer value is loaded into the buffer and held there. After the CPU reads the EPA*x*_TIME register, the contents of the capture buffer are automatically transferred into EPA*x*_TIME and the EPA interrupt pending bit is set.

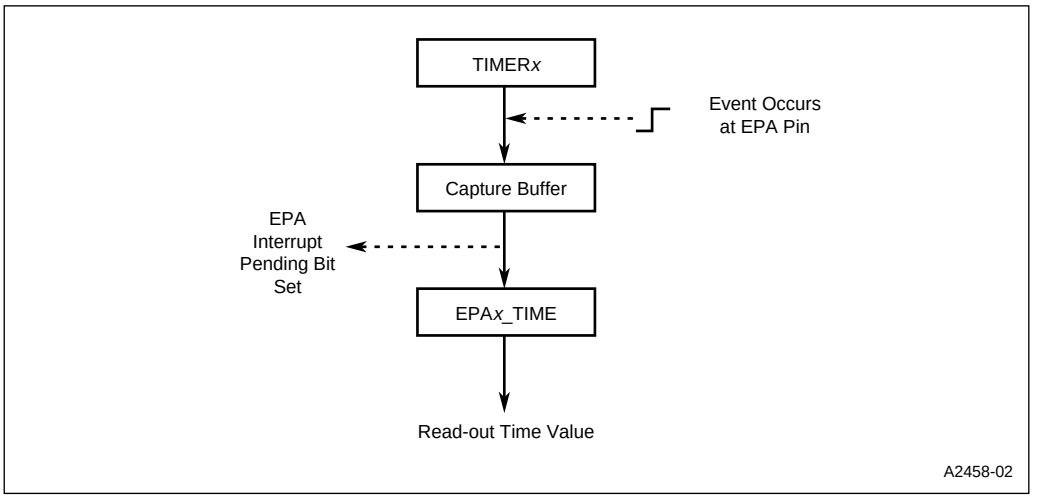


Figure 10-6. EPA Simplified Input-capture Structure

If a third event occurs before the CPU reads the event-time register, the overwrite bit (EPA*x*_CON.0) determines how the EPA will handle the event. If the bit is clear, the EPA ignores the third event. If the bit is set, the third event time overwrites the second event time in the capture buffer. Both situations set the overrun interrupt pending bit, and if the interrupt is enabled, they generate an overrun interrupt. Table 10-4 summarizes the possible actions when a valid event occurs.

NOTE

In order for an event to be captured, the signal must be stable for at least two state times both before and after the transition occurs (Figure 10-7).

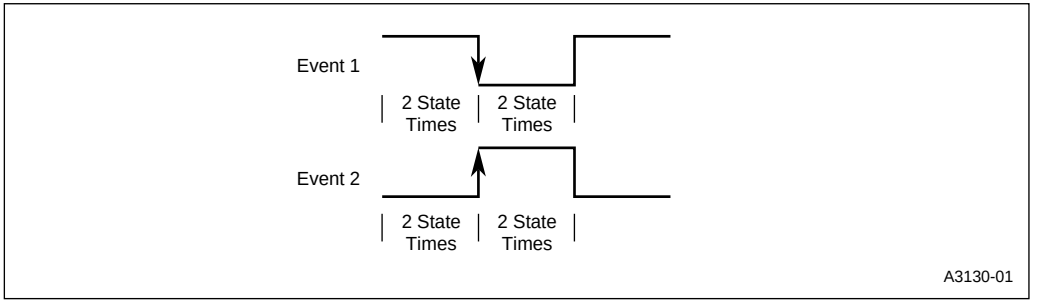


Figure 10-7. Valid EPA Input Events

Table 10-4. Action Taken when a Valid Edge Occurs

Overwrite Bit (EPA*x*_CON.0)	Status of Capture Buffer & EPA*x*_TIME	Action taken when a valid edge occurs
0	empty	Edge is captured and event time is loaded into the capture buffer and EPA*x*_TIME register.
0	full	New data is ignored — no capture, EPA interrupt, or transfer occurs; OVR*x* interrupt pending bit is set.
1	empty	Edge is captured and event time is loaded into the capture buffer and EPA*x*_TIME register.
1	full	Old data is overwritten in the capture buffer; OVR*x* interrupt pending bit is set.

An input capture event does not set the interrupt pending bit until the captured time value actually moves from the capture buffer into the EPA*x*_TIME register. If the buffer contains data and the PTS is used to service the interrupts, then two PTS interrupts occur almost back-to-back (that is, with one instruction executed between the interrupts).

10.4.1.1 EPA Overruns

Overruns occur when an EPA input transitions at a rate that cannot be handled by the EPA interrupt service routine. If no overrun handling strategy is in place, and if the following three conditions exist, a situation may occur where both the capture buffer and the EPA*x*_TIME register contain data, and no EPA interrupt is generated.

- an input signal with a frequency high enough to cause overruns is present on an enabled EPA pin, and
- the overwrite bit is set (EPA*x*_CON.0 = 1; old data is overwritten on overrun), and
- the EPA*x*_TIME register is read at the exact instant that the EPA recognizes the captured edge as valid.

The input frequency at which this occurs depends on the length of the interrupt service routine as well as other factors. Unless the interrupt service routine includes a check for overruns, this situation will remain the same until the device is reset or the EPA*x*_TIME register is read. The act of reading EPA*x*_TIME allows the buffered time value to be moved into EPA*x*_TIME. This clears the buffer and allows another event to be captured. Remember that the act of the transferring the buffer contents to the EPA*x*_TIME register is what actually sets the EPA*x* interrupt pending bit and generates the interrupt.

10.4.1.2 Preventing EPA Overruns

Any one of the following methods can be used to prevent or recover from an EPA overrun situation.

- Clear EPA*x*_CON.0

  When the overwrite bit (EPA*x*_CON.0) is zero, the EPA does not consider the captured edge until the EPA*x*_TIME register is read and the data in the capture buffer is transferred to EPA*x*_TIME. This prevents the situation by ignoring new input capture events when both the capture buffer and EPA*x*_TIME contain valid capture times. The OVR*x* pending bit in EPA_PEND is set to indicate that an overrun occurred.

- Enable the OVR*x* interrupt and read the EPA*x*_TIME register within the ISR

  If this situation occurs, the overrun (OVR*x*) interrupt will be generated. The OVR*x* interrupt will then be acknowledged and its interrupt service routine will read the EPA*x*_TIME register. After the CPU reads the EPA*x*_TIME register, the buffered data moves from the buffer to the EPA*x*_TIME register. This sets the EPA interrupt pending bit.

10.4.2 Operating in Compare Mode

When the selected timer value matches the event-time value, the action specified in the control register occurs (i.e., the pin is set, cleared, or toggled). If the re-enable bit (EPA*x*_CON.3) is set, the action reoccurs on every timer match. If the re-enable bit is cleared, the action does not reoccur until a new value is written to the event-time register. See "Programming the Capture/Compare Channels" on page 10-18 for configuration information.

In compare mode, you can use the EPA to produce a pulse-width modulated (PWM) output. The following sections describe four possible methods.

10.4.2.1 Generating a Low-speed PWM Output

You can generate a low-speed, pulse-width modulated output with a single EPA channel and a standard interrupt service routine. Configure the EPA channel as follows: compare mode, toggle output, and the compare function re-enabled. Select standard interrupt service, enable the EPA interrupt, and globally enable interrupts with the EI instruction. When the assigned timer/counter value matches the value in the event-time register, the EPA toggles the output pin and generates an interrupt. The interrupt service routine loads a new value into EPA*x*_TIME.

The maximum output frequency depends upon the total interrupt latency and the interrupt-service execution times used by your system. As additional EPA channels and the other functions of the microcontroller are used, the maximum PWM frequency decreases because the total interrupt latency and interrupt-service execution time increases. To determine the maximum, low-speed PWM frequency in your system, calculate your system's worst-case interrupt latency and worst-case interrupt-service execution time, and then add them together. The worst-case interrupt latency is the total latency of all the interrupts (both normal and PTS) used in your system. The worst-case interrupt-service execution time is the total execution time of all interrupt service routines and PTS routines.

Assume a system with a single EPA channel, a single enabled interrupt, and the following interrupt service routine.

```
;If EPA0-3 interrupt is generated
EPA0-3_ISR:
     PUSHA
     LD EPAx_CON, #toggle_command
     ADD EPAx_TIME, TIMERx, [next_duty_ptr]; Load next event time
     POPA
     RET
```

The worst-case interrupt latency for a single-interrupt system is 56 state times for external stack usage and 54 state times for internal stack usage (see “Standard Interrupt Latency” on page 6-8). To determine the execution time for an interrupt service routine, add up the execution time of the instructions (Table A-9).

The total execution time for the ISR that services interrupts EPA3:0 is 79 state times for external stack usage or 71 state times for internal stack usage. Therefore, a single capture/compare channel 0–3 can be updated every 125 state times assuming internal stack usage (54 + 71). Each PWM period requires two updates (one setting and one clearing), so the execution time for a PWM period equals 250 state times. When the input frequency on XTAL1 is 25 MHz and the phase-locked loop is disabled on the 80C196NU, the PWM period is 20 µs and the maximum PWM frequency is 50 kHz.

10.4.2.2 Generating a Medium-speed PWM Output

You can generate a medium-speed, pulse-width modulated output with a single EPA channel and the PTS set up in PWM toggle mode. “PWM Toggle Mode Example” on page 6-27 describes how to configure the EPA and PTS. Once started, this method requires no CPU intervention unless you need to change the output frequency. The method uses a single timer/counter. The timer/counter is not interrupted during this process, so other EPA channels can also use it if they do not reset it.

The maximum output frequency depends upon the total interrupt latency and interrupt-service execution time. As additional EPA channels and the other functions of the microcontroller are used, the maximum PWM frequency decreases because the total interrupt latency and interrupt-service execution time increases. To determine the maximum, medium-speed PWM frequency in your system, calculate your system's worst-case interrupt latency and worst-case interrupt-service execution time, and then add them together. The worst-case interrupt latency is the total latency of all the interrupts (both normal and PTS) used in your system. The worst-case interrupt-service execution time is the total execution time of all interrupt service routines and PTS cycles.

Assume a system with a single EPA channel, a single enabled interrupt, and PTS service. Also assume that the PTS is initialized and that the duty cycle and frequency are fixed. The worst-case interrupt latency for a single-interrupt system with PTS service is 43 state times (see "PTS Interrupt Latency" on page 6-9). The PTS cycle execution time in PWM toggle mode is 15 state times (Table 6-4 on page 6-10). Therefore, a single capture/compare channel can be updated every 58 state times (43 + 15). Each PWM period requires two updates (one setting and one clearing), so the execution time for a PWM period equals 116 state times. When the input frequency on XTAL1 is 25 MHz and the phase-locked loop is disabled on the 80C196NU, the PWM period is 9.27 µs and the maximum PWM frequency is 107.8 kHz.

10.4.2.3 Generating a High-speed PWM Output

You can generate a high-speed, pulse-width modulated output with a pair of EPA channels and the PTS set up in PWM remap mode. "PWM Remap Mode Example" on page 6-32 describes how to configure the EPA and PTS. The remap bit (bit 8) must be set in EPA1_CON (to pair EPA0 and EPA1) or EPA3_CON (to pair EPA2 and EPA3). One channel must be configured to set the output; the other, to clear it. At the set (or clear) time, the PTS reads the old time value from EPA*x*_TIME, adds to it the PWM period constant, and returns the new value to EPA*x*_TIME. Set and clear times can be programmed to differ by as little as one timer count, resulting in very narrow pulses. Once started, this method requires no CPU intervention unless you need to change the output frequency. The method uses a single timer/counter. The timer/counter is not interrupted during this process, so other EPA channels can also use it if they do not reset it.

To determine the maximum, high-speed PWM frequency in your system, calculate your system's worst-case interrupt latency and then double it. The worst-case interrupt latency is the total latency of all the interrupts (both normal and PTS) used in your system.

Assume a system that uses a pair of remapped EPA channels (i.e., EPA0 and 1 or EPA3 and 4), two enabled interrupts, and PTS service. Also assume that the PTS is initialized and that the duty cycle and frequency are fixed. The worst-case interrupt latency for a single-interrupt system with PTS service is 43 state times (see "PTS Interrupt Latency" on page 6-9). In this mode, the maximum period equals twice the PTS latency. Therefore, the execution time for a PWM period equals 86 state times. When the input frequency on XTAL1 is 25 MHz and the phase-locked loop is disabled on the 80C196NU, the PWM period is 6.88 µs and the maximum PWM frequency is 145.3 kHz.

10.4.2.4 Generating the Highest-speed PWM Output

You can generate a highest-speed, pulse-width modulated output with a pair of EPA channels and a dedicated timer/counter. The first channel toggles the output when the timer value matches EPA*x*_TIME, and at some later time, the second channel toggles the output again **and** resets the timer/counter. This restarts the cycle. No interrupts are required, resulting in the highest possible speed. Software must calculate and load the appropriate EPA*x*_TIME values and load them at the correct time in the cycle in order to change the frequency or duty cycle.

With this method, the resolution of the EPA (selected by the T*x*CONTROL registers; see Figure 10-8 on page 10-16 and Figure 10-9 on page 10-17) determines the maximum PWM output frequency. (Resolution is the minimum time required between consecutive captures or compares.) When the input frequency on XTAL1 is 25 MHz and the phase-locked loop is disabled on the 80C196NU, a 160 ns resolution results in a maximum PWM of 6.25 MHz.

10.5 PROGRAMMING THE EPA AND TIMER/COUNTERS

This section discusses configuring the port pins for the EPA and the timer/counters; describes how to program the timers and the capture/compare channels; and explains how to enable the EPA interrupts.

10.5.1 Configuring the EPA and Timer/Counter Port Pins

Before you can use the EPA, you must configure the pins of port 1 to serve as the special-function signals for the EPA and, optionally, for the timer/counter clock source and direction control signals. See “Bidirectional Ports 1–4” on page 7-1 for information about configuring the port pins.

NOTE

If you use T2CLK as the timer 2 input clock, you cannot use EPA capture/compare channel 0. If you use T2DIR as the timer 2 direction-control source, you cannot use EPA capture/compare channel 1.

Table 10-1 on page 10-2 lists the pins associated with the EPA and the timer/counters. Pins that are not being used for an EPA channel or timer/counter can be configured as standard I/O.

10.5.2 Programming the Timers

The control registers for the timers are T1CONTROL (Figure 10-8) and T2CONTROL (Figure 10-9). Write to these registers to configure the timers. Write to the TIMER1 and TIMER2 registers (see Table 10-2 on page 10-3 for addresses) to load a specific timer value.

T1CONTROL Address: 1F90H
Reset State: 00H

The timer 1 control (T1CONTROL) register determines the clock source, counting direction, and count rate for timer 1.

7							0
CE	UD	M2	M1	M0	P2	P1	P0

Bit Number	Bit Mnemonic	Function
7	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disables timer 1 = enables timer
6	UD	Up/Down This bit determines the timer counting direction, in selected modes (see mode bits, M2:0). 0 = count down 1 = count up
5:3	M2:0	EPA Clock Direction Mode Bits These bits determine the timer clocking source and direction control source. **M2 M1 M0 Clock Source Direction Source** 0 0 0 f/4 UD bit (T1CONTROL.6) X 0 1 T1CLK pin† UD bit (T1CONTROL.6) 0 1 0 f/4 T1DIR pin 0 1 1 T1CLK pin† T1DIR pin 1 1 1 quadrature clocking using T1CLK and T1DIR † If an external clock is selected, the timer counts on both the rising **and** falling edges of the clock.
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. **P2 P1 P0 Prescaler Divisor Resolution†** 0 0 0 divide by 1 (disabled) 160 ns 0 0 1 divide by 2 320 ns 0 1 0 divide by 4 640 ns 0 1 1 divide by 8 1.28 µs 1 0 0 divide by 16 2.56 µs 1 0 1 divide by 32 5.12 µs 1 1 0 divide by 64 10.24 µs 1 1 1 divide by 128 (NU only) 20.48 µs † At f = 25 MHz. Use the formula on page 10-6 to calculate the resolution at other frequencies.

Figure 10-8. Timer 1 Control (T1CONTROL) Register

T2CONTROL Address: 1F94H
Reset State: 00H

The timer 2 control (T2CONTROL) register determines the clock source, counting direction, and count rate for timer 2.

7							0
CE	UD	M2	M1	M0	P2	P1	P0

Bit Number	Bit Mnemonic	Function
7	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disables timer 1 = enables timer
6	UD	Up/Down This bit determines the timer counting direction, in selected modes (see mode bits, M2:0). 0 = count down 1 = count up
5:3	M2:0	EPA Clock Direction Mode Bits. These bits determine the timer clocking source and direction source **M2 M1 M0 Clock Source Direction Source** 0 0 0 f/4 UD bit (T2CONTROL.6) X 0 1 T2CLK pin† UD bit (T2CONTROL.6) 0 1 0 f/4 T2DIR pin 0 1 1 T2CLK pin† T2DIR pin 1 0 0 timer 1 overflow UD bit (T2CONTROL.6) 1 1 0 timer 1 same as timer 1 1 1 1 quadrature clocking using T2CLK and T2DIR † If an external clock is selected, the timer counts on both the rising **and** falling edges of the clock.
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. **P2 P1 P0 Prescaler Resolution†** 0 0 0 divide by 1 (disabled) 160 ns 0 0 1 divide by 2 320 ns 0 1 0 divide by 4 640 ns 0 1 1 divide by 8 1.28 µs 1 0 0 divide by 16 2.56 µs 1 0 1 divide by 32 5.12 µs 1 1 0 divide by 64 10.24 µs 1 1 1 divide by 128 (NU only) 20.48 µs † At f = 25 MHz. Use the formula on page 10-6 to calculate the resolution at other frequencies.

Figure 10-9. Timer 2 Control (T2CONTROL) Register

10.5.3 Programming the Capture/Compare Channels

The EPA*x*_CON register controls the function of its assigned capture/compare channel. The registers for EPA0 and EPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit (RM), which is used to enable and disable remapping for high-speed PWM generation (see "Generating a High-speed PWM Output" on page 10-14). This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON **must** be addressed as **words**, while the others can be addressed as bytes.

To program a compare event, write to EPA*x*_CON (Figure 10-10) to configure the EPA capture/compare channel and then load the event time into EPA*x*_TIME. To program a capture event, you need only write to EPA*x*_CON. Table 10-5 shows the effects of various combinations of EPA*x*_CON bit settings.

Table 10-5. Example Control Register Settings and EPA Operations

Capture Mode								
TB	**CE**	**MODE**		**RE**	**—**	**ROT**	**ON/RT**	**Operation**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
X	0	0	0	—	0	—	0	None
X	0	0	1	—	0	X	X	Capture on falling edges
X	0	1	0	—	0	X	X	Capture on rising edges
X	0	1	1	—	0	X	X	Capture on both edges
X	0	X	1	—	0	1	X	Reset opposite timer
X	0	1	X	—	0	1	X	Reset opposite timer
Compare Mode								
TB	**CE**	**MODE**		**RE**	**—**	**ROT**	**ON/RT**	**Operation**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
X	1	0	0	X	0	—	0	None
X	1	0	1	X	0	X	X	Clear output pin
X	1	1	0	X	0	X	X	Set output pin
X	1	1	1	X	0	X	X	Toggle output pin
X	1	X	X	X	0	0	1	Reset same timer
X	1	X	X	X	0	1	1	Reset opposite timer

NOTES: — = bit is not used
X = bit may be used, but has no effect on the described operation. These bits cause other operations to occur.

EPA*x*_CON
***x* = 0–3**

Address: Table 10-2 on page 10-3
Reset State: 00H

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels. The registers for EPA0 andEPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit. This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON must be addressed as words, while the others can be addressed as bytes.

***x* = 1, 3**

15							8
—	—	—	—	—	—	—	RM
7							**0**
TB	CE	M1	M0	RE	—	ROT	ON/RT

***x* = 0, 2**

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

Bit Number	Bit Mnemonic	Function
15:9†	—	Reserved; always write as zeros.
8†	RM	Remap Feature The remap feature applies to the compare mode of the EPA1 and EPA3 only. When the remap feature of EPA1 is enabled, EPA capture/compare channel 0 shares output pin EPA1 with EPA capture/compare channel 1. When the remap feature of EPA3 is enabled, EPA capture/compare channel 2 shares output pin EPA3 with EPA capture/compare channel 3. 0 = remap feature disabled 1 = remap feature enabled
7	TB	Time Base Select Specifies the reference timer. 0 = timer 1 is the reference timer and timer 2 is the opposite timer 1 = timer 2 is the reference timer and timer 1 is the opposite timer A compare event (clearing, setting, or toggling an output pin; and/or resetting either timer) occurs when the reference timer matches the time programmed in the event-time register. When a capture event (falling edge, rising edge, or an edge change on the EPA*x* pin) occurs, the reference timer value is saved in the EPA event-time register (EPA*x*_TIME).
6	CE	Compare Enable Determines whether the EPA channel operates in capture or compare mode. 0 = capture mode 1 = compare mode

† These bits apply to the EPA1_CON and EPA3_CON registers only.

Figure 10-10. EPA Control (EPA*x*_CON) Registers

EPA*x*_CON (Continued) Address: Table 10-2 on page 10-3
***x* = 0–3** Reset State: 00H

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels. The registers for EPA0 andEPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit. This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON must be addressed as words, while the others can be addressed as bytes.

***x* = 1, 3**

15							8
—	—	—	—	—	—	—	RM
7							**0**
TB	CE	M1	M0	RE	—	ROT	ON/RT

***x* = 0, 2**

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

Bit Number	Bit Mnemonic	Function
5:4	M1:0	EPA Mode Select In capture mode, specifies the type of event that triggers an input capture. In compare mode, specifies the action that the EPA executes when the reference timer matches the event time. **M1 M0 Capture Mode Event** 0 0 no capture 0 1 capture on falling edge 1 0 capture on rising edge 1 1 capture on either edge **M1 M0 Compare Mode Action** 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin
3	RE	Re-enable Re-enable applies to the compare mode only. It allows a compare event to continue to execute each time the event-time register (EPA*x*_TIME) matches the reference timer rather than only upon the first time match. 0 = compare function is disabled after a single event 1 = compare function always enabled
2	—	Reserved; always write as zero.

† These bits apply to the EPA1_CON and EPA3_CON registers only.

Figure 10-10. EPA Control (EPA*x*_CON) Registers (Continued)

EPA*x*_CON (Continued) Address: Table 10-2 on page 10-3
***x* = 0–3** Reset State: 00H

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels. The registers for EPA0 andEPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit. This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON must be addressed as words, while the others can be addressed as bytes.

***x* = 1, 3**

15							8
—	—	—	—	—	—	—	RM
7							**0**
TB	CE	M1	M0	RE	—	ROT	ON/RT

***x* = 0, 2**

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

Bit Number	Bit Mnemonic	Function
1	ROT	Reset Opposite Timer Controls different functions for capture and compare modes. **In Capture Mode:** 0 = causes no action 1 = resets the opposite timer **In Compare Mode:** Selects the timer that is to be reset if the RT bit is set. 0 = selects the reference timer for possible reset 1 = selects the opposite timer for possible reset The TB bit (bit 7) selects which is the reference timer and which is the opposite timer.
0	ON/RT	Overwrite New/Reset Timer The ON/RT bit functions as overwrite new in capture mode and reset timer in compare mode. **In Capture Mode (ON):** An overrun error is generated when an input capture occurs while the event-time register (EPA*x*_TIME) and its buffer are both full. When an overrun occurs, the ON bit determines whether old data is overwritten or new data is ignored: 0 = ignores new data 1 = overwrites old data in the buffer **In Compare Mode (RT):** 0 = disables the reset function 1 = resets the ROT-selected timer

† These bits apply to the EPA1_CON and EPA3_CON registers only.

Figure 10-10. EPA Control (EPA*x*_CON) Registers (Continued)

10.6 ENABLING THE EPA INTERRUPTS

The EPA generates four individual event interrupts, EPA3:0, from the four capture/compare channels and two timer interrupts, OVRTM1 and OVRTM2, from timer 1 and timer 2. These interrupts are directly mapped into the two 8-bit interrupt pending registers (INT_PEND and INT_PEND1). The four separate capture overrun interrupts from EPA3:0 are multiplexed and mapped into two bits in INT_PEND1. The capture overrun interrupts from EPA0 and EPA1 are multiplexed and mapped into OVR0_1 (bit 4) of INT_PEND1; the capture overrun interrupts from EPA2 and EPA3 are multiplexed and mapped into OVR2_3 (bit 5) of INT_PEND1. To enable the interrupts, set the corresponding bits in the the two 8-bit interrupt mask registers (INT_MASK and INT_MASK1). To enable the individual sources of the capture overrun interrupts OVR0_1 and OVR2_3, set the corresponding bits in the EPA mask register (EPA_MASK). (Chapter 6, "Standard and PTS Interrupts," discusses the interrupts in greater detail.)

EPA_MASK

Address: 1F9CH
Reset State: AAH

The EPA interrupt mask (EPA_MASK) register enables or disables (masks) the multiplexed EPA3:0 overrun interrupts (OVR3:0).

7							0
—	OVR3	—	OVR2	—	OVR1	—	OVR0

Bit Number	Bit Mnemonic	Function
7, 5, 3, 1	—	Reserved; for compatibility with future devices, write zeros to these bits.
6, 4, 2, 0	OVR3 OVR2 OVR1 OVR0	Setting this bit enables the corresponding source as a shared overrun interrupt source. The shared overrun interrupts (OVR0_1 and OVR2_3) are enabled by setting their interrupt enable bits in the interrupt mask 1 (INT_MASK1) register.

Figure 10-11. EPA Interrupt Mask (EPA_MASK) Register

10.7 DETERMINING EVENT STATUS

In compare mode, an interrupt pending bit is set each time a match occurs on an enabled event (even if the interrupt is specifically masked in the mask register). In capture mode, an interrupt pending bit is set each time a programmed event is captured and the event time moves from the capture buffer to the EPA*x*_TIME register. If the capture buffer is full when an event occurs, an overrun interrupt pending bit is set.

Timer overflows and capture overruns also set interrupt pending bits. You can mask the interrupts by clearing bits in EPA_MASK (Figure 10-11), INT_MASK, and INT_MASK1. If an interrupt is masked, software can still poll the interrupt pending registers to determine whether an event has occurred.

EPA_PEND† Address: 1F9EH
Reset State: AAH

When hardware detects a pending EPA3:0 overrun interrupt (OVR3:0), it sets the corresponding bit in the EPA interrupt pending (EPA_PEND) register. OVR0 and OVR1 are multiplexed to share one bit (OVR0_1) in the INT_PEND1 register. Similarly, OVR2 and OVR3 are multiplexed to share another bit (OVR2_3) in the INT_PEND1 register.

7							0
—	OVR3	—	OVR2	—	OVR1	—	OVR0

Bit Number	Function
7, 5, 3, 1	Reserved. These bits are undefined.
6, 4, 2, 0	Any set bit indicates that the corresponding overrun interrupt source is pending.

† This register was called EPA_STAT in previous documentation for the 8XC196NP.

Figure 10-12. EPA Interrupt Pending (EPA_PEND) Register

The EPA interrupt pending register, EPA_PEND, has the same bit structure as the EPA_MASK register. EPA_PEND is similar to an interrupt pending register in that it shows the status of the individual capture/compare overrun interrupts. The bits in EPA_PEND can be polled to determine the exact source of an OVR0_1 or OVR2_3 interrupt. However, hardware does not clear status bits in this register when it vectors to the interrupt service routine for an interrupt pair (OVR0_1, OVR2_3) so the user's code must clear the register. Instead it clears the OVR0_1 or OVR2_3 bit in the INT_MASK register. Also, software cannot generate an interrupt by setting a bit in EPA_PEND.

10.7.1 Using Software to Service the Multiplexed Overrun Interrupts

The multiplexed overrun interrupts should normally be serviced by interrupt service routines because the PTS cannot determine the exact source of the interrupt. When an OVR0_1 or OVR2_3 occurs, the user's software service routine can poll the bits of the EPA_PEND register, which has a bit for each overrun source, to determine which of the four capture/compare channels caused the interrupt. The individual sources can be masked by bits in the EPA_MASK register.

10.8 PROGRAMMING EXAMPLES FOR EPA CHANNELS

The three programming examples provided in this section demonstrate the use of the EPA channel for a compare event, for a capture event, and for generation of a PWM signal. The programs demonstrate the detection of events by a polling scheme, by interrupts, and by the PTS. All three examples were created using *Ap*BUILDER, an interactive application program available through Intel Literature Fulfillment. These sample program were written in the C programming language. ASM versions are also available from *Ap*BUILDER.

NOTE

The initialization file (80c196np.h) used in these examples is available from the Intel Applications BBS.

10.8.1 EPA Compare Event Program

This example C program demonstrates an EPA compare event. It sets up EPA channel 0 to toggle its output pin whenever timer 1 is zero. This program uses no interrupts; a polling scheme detects the EPA event. The program initializes EPA channel 0 for a compare event.

```
#pragma model(EX)
#include <80c196np.h>

#define   COMPARE      0x40
#define   RE_ENABLE    0x08
#define   TOGGLE_PIN   0x30
#define   USE_TIMER1   0x00
#define   EPA0_INT_BIT    7

void init_epa0()
{
 epa0_con =    COMPARE |
               TOGGLE_PIN |
               RE_ENABLE |
               USE_TIMER1;
 epa0_time = 0;
 setbit(p1_reg, 0); /* int reg */
 clrbit(p1_dir, 0); /* make output pin */
 setbit(p1_mode, 0);/* select EPA mode */
}

void init_timer1()
{
 t1control =   COUNT_ENABLE |
               COUNT_UP |
               CLOCK_INTERNAL |
               DIVIDE_BY_1;
}
```

```
void poll_epa0()
{
 if(checkbit(int_pend, EPA0_INT_BIT))
     {
     /*  Insert user code for event channel 0 here. */
     /*  Since this event is absolute and re-enabled, no polling is neccessary.*/
     clrbit(int_pend, EPA0_INT_BIT);
     }
}

void main(void)
{
/* Initialize the timers before using the epa */
init_timer1();
init_epa0();
/*  EPA events can be serviced by polling int_pend or epa_pend. */
while(1)
     {
     poll_epa0();
     }
}
```

10.8.2 EPA Capture Event Program

This example C program demonstrates an EPA capture event. It sets up EPA channel 0 to capture edges (rising and falling) on the EPA0 pin. The program also shows how to set up an the EPA interrupt. You can add your own code for the interrupt service routine.

```
#pragma model(EX)
#include <80c196np.h>

#define  COUNT_ENABLE         0x80
#define  COUNT_UP             0x40
#define  CLOCK_INTERNAL       0x00
#define  DIVIDE_BY_1          0x00
#define  CAPTURE              0x00
#define  BOTH_EDGE            0x30
#define  USE_TIMER1           0x00
#define  EPA0_INT_BIT         7

void init_epa0()
{
 epa0_con = CAPTURE  ¦
               BOTH_EDGE  ¦
               USE_TIMER1;
 setbit(p1_reg, 0); /*  int reg  */
 setbit(p1_dir, 0); /*  make input pin  */
 setbit(p1_mode, 0);/*  select EPA mode  */
 setbit(int_mask, EPA0_INT_BIT);/*  unmask EPA interrupts  */
}

#pragma interrupt(epa0_interrupt=EPA0_INT_BIT)
void epa0_interrupt()
{
 unsigned int time_value;
```

```
 time_value = epa0_time; /* must read to prevent overrun */
}
void init_timer1()
{
 t1control =   COUNT_ENABLE |
               COUNT_UP |
               CLOCK_INTERNAL |
               DIVIDE_BY_1;
}

void main(void)
{
 unsigned int time_value;

 /* Initialize the timers and interrupts before using the EPA */
 init_timer1();
 init_epa0();
 enable();     /*  Globally enable interrupts  */
 while(1);     /*  loop forever, wait for interrupts to occur */
}
```

10.8.3 EPA PWM Output Program

This example C program demonstrates the generation of a PWM signal using the EPA's PWM toggle mode (see "PWM Modes" on page 6-26) and shows how to service the interrupts with the PTS. The PWM signal in this example has a 50% duty cycle.

```
#pragma model(EX)
#include <80c196np.h>
#define   PTS_BLOCK_BASE      0x98

/* Create typedef template for the PWM_TOGGLE mode control block.*/
typedef struct PWM_toggle_ptscb_t {
          unsigned char  unused;
          unsigned char  ptscon;
          void  *pts_ptr;
          unsigned int constant1;
          unsigned int constant2;
          } PWM_toggle_ptscb;

/* This locates the PTS block mode control block in register ram. This */
/* control block may be located at any quad-word boundary. */

register PWM_toggle_ptscb PWM_toggle_CB_3;
#pragma locate(PWM_toggle_CB_3=PTS_BLOCK_BASE)

/* The PTS vector must contain the address of the PTS control block.*/
#pragma pts(PWM_toggle_CB_3=0x3)




/* Sample PTS control block initialization sequence.*/
```

```
void Init_PWM_toggle_PTS3(void)
{
    disable();          /* disable all interrupts */
    disable_pts();      /* disable the PTS interrupts */

    PWM_toggle_CB_3.constant2  = 127;
    PWM_toggle_CB_3.constant1  = 127;
    PWM_toggle_CB_3.pts_ptr    = (void *)&EPA0_TIME;
    PWM_toggle_CB_3.ptscon     = 0x42;

/* Sample code that could be used to generate a PWM with an EPA channel.*/

    setbit(p1_reg, 0x1);  /*  init output    */
    clrbit(p1_dir, 0x1);  /*  set to output  */
    setbit(p1_mode, 0x1); /*  set special function*/
    setbit(ptssel, 0x8);
    setbit(int_mask, 0x0)
}

void main(void)
{
 Init_PWM_toggle_PTS3();
 epa1_con = 0x78;        /* toggle, timer1, compare, re-enable */
 epa1_timer = 127;
 t1control = 0xC2;       /* enable timer, up 1 microsecond @ 16 MHz  */
 enable_pts();
 while(1);
}
```

11

Minimum Hardware Considerations

CHAPTER 11
MINIMUM HARDWARE CONSIDERATIONS

The 8XC196NP and 80C196NU have several basic requirements for operation within a system. This chapter describes options for providing the basic requirements and discusses other hardware considerations.

11.1 MINIMUM CONNECTIONS

Table 11-1 lists the signals that are required for the device to function and Figure 11-1 shows the connections for a minimum configuration.

Table 11-1. Minimum Required Signals

Signal Name	Type	Description
RESET#	I/O	Reset A level-sensitive reset input to and open-drain system reset output from the micro-controller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown, standby, and idle modes, asserting RESET# causes the chip to reset and return to normal operating mode. After a device reset, the first instruction fetch is from FF2080H (or F2080H in external memory). For the 80C196NP and 80C196NU, the program and special-purpose memory locations (FF2000–FF2FFFH) reside in external memory. For the 83C196NP, these locations can reside either in external memory or in internal ROM.
RPD	I	Return from Powerdown Timing pin for the return-from-powerdown circuit. If your application uses powerdown mode, connect a capacitor† between RPD and V_{SS} if **either** of the following conditions is true. • the internal oscillator is the clock source • the phase-locked loop (PLL) circuitry (80C196NU only) is enabled (see PLLEN2:1 signal description) The capacitor causes a delay that enables the oscillator and PLL circuitry to stabilize before the internal CPU and peripheral clocks are enabled. The capacitor is not required if your application uses powerdown mode and if **both** of the following conditions are true. • an external clock input is the clock source • the phase-locked loop circuitry (80C196NU only) is disabled If your application does not use powerdown mode, leave this pin unconnected. † Calculate the value of the capacitor using the formula found on page 12-11.
V_{CC}	PWR	Digital Supply Voltage Connect each V_{CC} pin to the digital supply voltage.
V_{SS}	GND	Digital Circuit Ground Connect each V_{SS} pin to ground through the lowest possible impedance path.

Table 11-1. Minimum Required Signals (Continued)

Signal Name	Type	Description
XTAL1	I	Input Crystal/Resonator or External Clock Input Input to the on-chip oscillator, internal phase-locked loop circuitry (80C196NU), and the internal clock generators. The internal clock generators provide the peripheral clocks, CPU clock, and CLKOUT signal. When using an external clock source instead of the on-chip oscillator, connect the clock input to XTAL1. The external clock signal must meet the V_{IH} specification for XTAL1 (see datasheet).
XTAL2	O	Inverted Output for the Crystal/Resonator Output of the on-chip oscillator inverter. Leave XTAL2 floating when the design uses a external clock source instead of the on-chip oscillator.

11.1.1 Unused Inputs

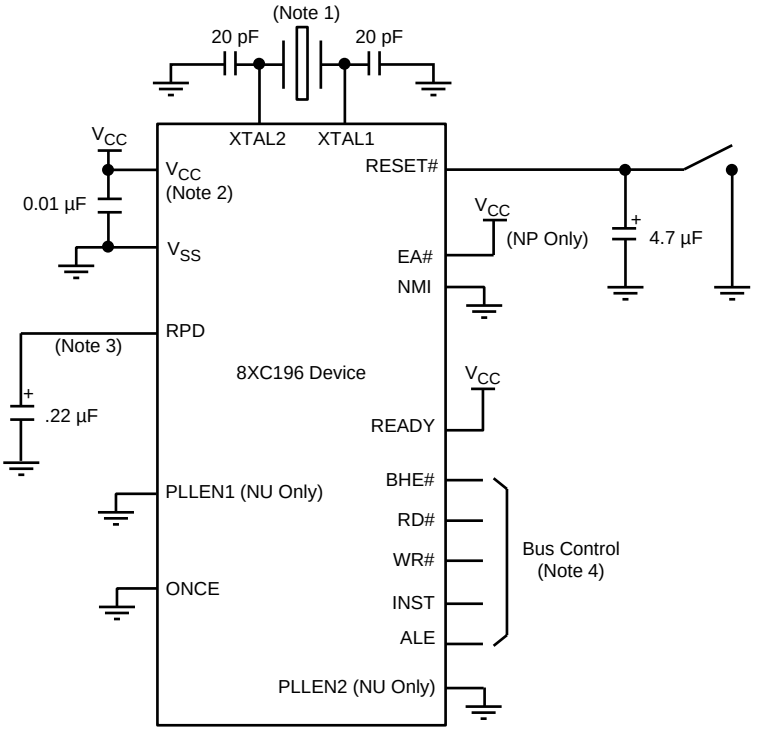
For predictable performance, it is important to tie unused inputs to V_{CC} or V_{SS}. Otherwise, they can float to a mid-voltage level and draw excessive current. Unused interrupt inputs may generate spurious interrupts if left unconnected.

11.1.2 I/O Port Pin Connections

Chapter 7, "I/O Ports," contains information about initializing and configuring the ports. Table 11-2 lists the sections, with page numbers, that contain the information for each port.

Table 11-2. I/O Port Configuration Guide

Port	Where to Find Configuration Information
Ports 1–4	"Bidirectional Port Pin Configurations" on page 7-7 and "Bidirectional Port Considerations" on page 7-9
EPORT	"Configuring EPORT Pins" on page 7-17



Notes:

1. See the datasheet for the oscillator frequency range (F_{OSC}) and the crystal manufacturer's datasheet for recommended load capacitors.
2. The number of V_{CC} and V_{SS} pins varies with package type (see datasheet). Be sure to connect all V_{CC} pins to the supply voltage and all V_{SS} pins to ground.
3. Connect the capacitor to RPD when using powerdown mode and the internal oscillator or phase-locked loop (NU only) circuitry. Otherwise, RPD may float.
4. No connection is required.

A2415-02

Figure 11-1. Minimum Hardware Connections

11.2 APPLYING AND REMOVING POWER

When power is first applied to the device, RESET# must remain continuously low for at least one state time after the power supply is within tolerance and the oscillator/clock has stabilized; otherwise, operation might be unpredictable. Similarly, when powering down a system, RESET# should be brought low before V_{CC} is removed; otherwise, an inadvertent write to an external location might occur. Carefully evaluate the possible effect of power-up and power-down sequences on a system.

11.3 NOISE PROTECTION TIPS

The fast rise and fall times of high-speed CMOS logic often produce noise spikes on the power supply lines and outputs. To minimize noise, it is important to follow good design and board layout techniques. We recommend liberal use of decoupling capacitors and transient absorbers. Add 0.01 µF bypass capacitors between V_{CC} and each V_{SS} pin to reduce noise (Figure 11-2). Place the capacitors as close to the device as possible. Use the shortest possible path to connect V_{SS} lines to ground and each other.

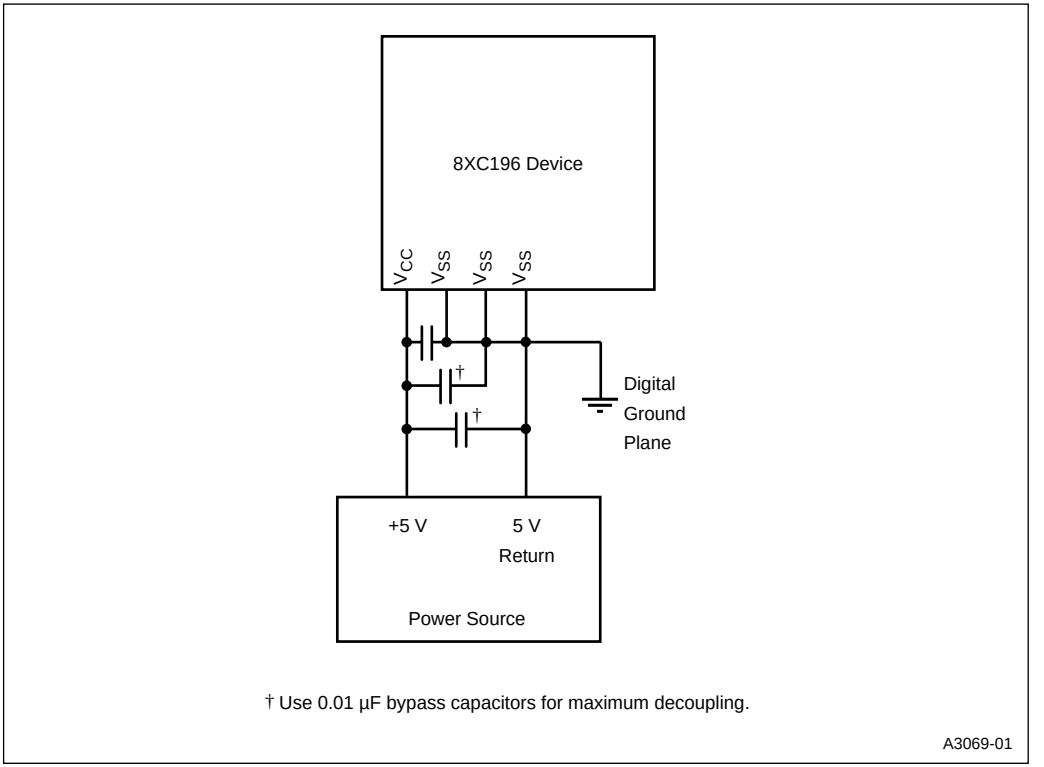


Figure 11-2. Power and Return Connections

Multilayer printed circuit boards with separate V_{CC} and ground planes also help to minimize noise. For more information on noise protection, refer to AP-125, *Designing Microcontroller Systems for Noisy Environments* and AP-711, *EMI Design Techniques for Microcontrollers in Automotive Applications*.

11.4 THE ON-CHIP OSCILLATOR CIRCUITRY

The on-chip oscillator circuit (Figure 11-3) consists of a crystal-controlled, positive reactance oscillator. In this application, the crystal operates in a parallel resonance mode. The feedback resistor, Rf, consists of paralleled *n*-channel and *p*-channel FETs controlled by the internal powerdown signal. In powerdown mode, Rf acts as an open and the output drivers are disabled, which disables the oscillator. Both the XTAL1 and XTAL2 pins have built-in electrostatic discharge (ESD) protection.

NOTE

For the 80C196NU, although the maximum external clock input frequency is 50 MHz, the maximum oscillator input frequency is limited to 25 MHz.

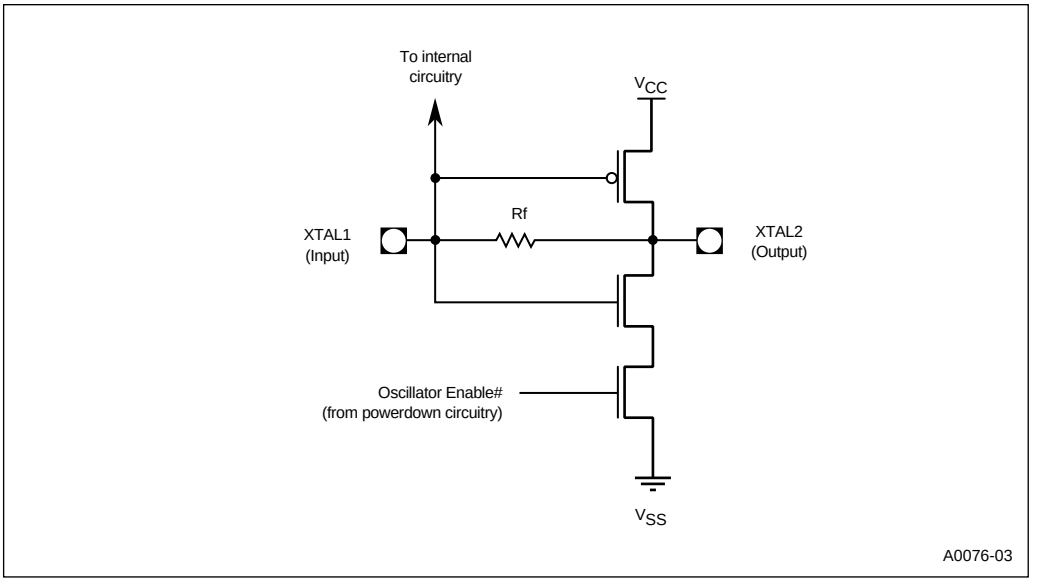
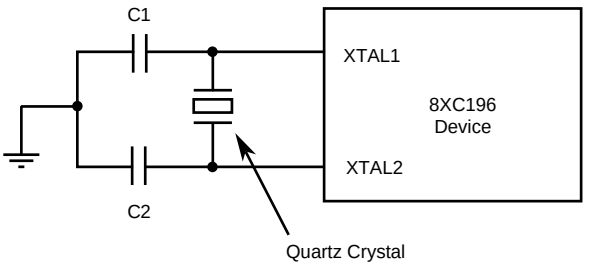


Figure 11-3. On-chip Oscillator Circuit

Figure 11-4 shows the connections between the external crystal and the device. When designing an external oscillator circuit, consider the effects of parasitic board capacitance, extended operating temperatures, and crystal specifications. Consult the manufacturer's datasheet for performance specifications and required capacitor values. With high-quality components, 20 pF load capacitors (C_L) are usually adequate for frequencies above 1 MHz.

Noise spikes on the XTAL1 or XTAL2 pin can cause a miscount in the internal clock-generating circuitry. Capacitive coupling between the crystal oscillator and traces carrying fast-rising digital signals can introduce noise spikes. To reduce this coupling, mount the crystal oscillator and capacitors near the device and use short, direct traces to connect to XTAL1, XTAL2, and V_{SS}. To further reduce the effects of noise, use grounded guard rings around the oscillator circuitry and ground the metallic crystal case.



Note:
Mount the crystal and capacitors close to the device using short, direct traces to XTAL1, XTAL2, and V_{SS}. When using a crystal, C1=C2≈20 pF. When using a ceramic resonator, consult the manufacturer for recommended oscillator circuitry.

A0273-02

Figure 11-4. External Crystal Connections

In cost-sensitive applications, you may choose to use a ceramic resonator instead of a crystal oscillator. Ceramic resonators may require slightly different load capacitor values and circuit configurations. Consult the manufacturer's datasheet for the requirements.

11.5 USING AN EXTERNAL CLOCK SOURCE

To use an external clock source, apply a clock signal to XTAL1 and let XTAL2 float (Figure 11-5). To ensure proper operation, the external clock source must meet the minimum high and low times (T_{XHXX} and T_{XLXX}) and the maximum rise and fall transition times (T_{XLXH} and T_{XHXL}) (Figure 11-6). The longer the rise and fall times, the higher the probability that external noise will affect the clock generator circuitry and cause unreliable operation. See the datasheet for required XTAL1 voltage drive levels and actual specifications.

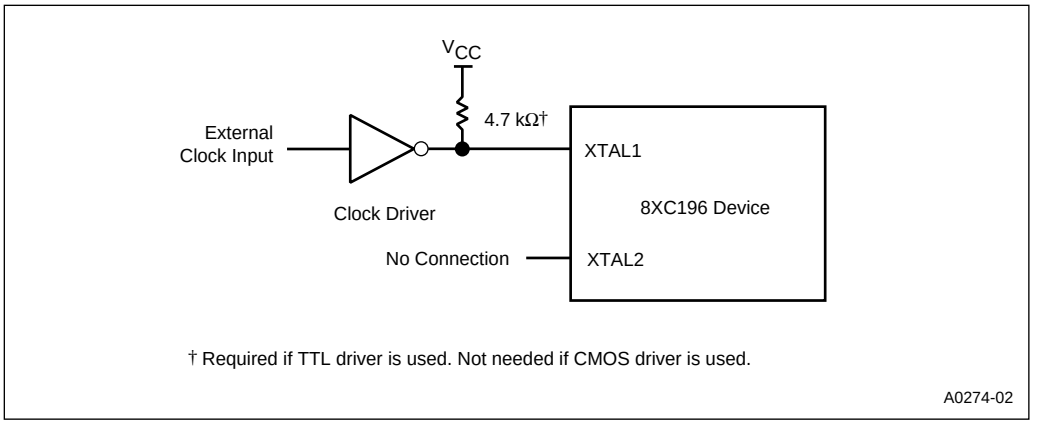


Figure 11-5. External Clock Connections

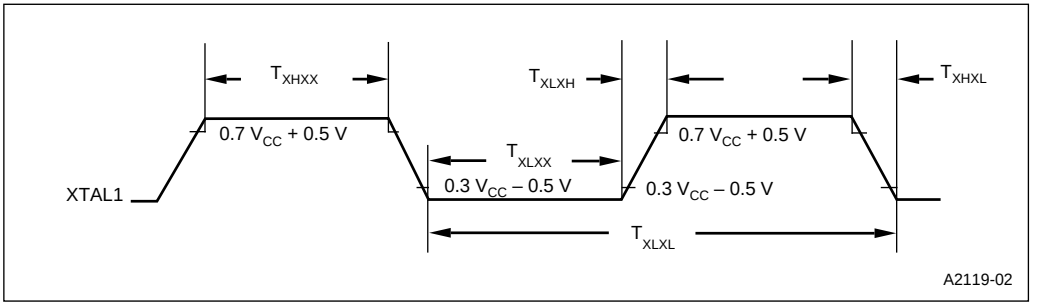


Figure 11-6. External Clock Drive Waveforms

At power-on, the interaction between the internal amplifier and its feedback capacitance (i.e., the Miller effect) may cause a load of up to 100 pF at the XTAL1 pin if the signal at XTAL1 is weak (such as might be the case during start-up of the external oscillator). This situation will go away when the XTAL1 input signal meets the V_{IL} and V_{IH} specifications (listed in the datasheet). If these specifications are met, the XTAL1 pin capacitance will not exceed 20 pF.

11.6 RESETTING THE DEVICE

Reset forces the device into a known state. As soon as RESET# is asserted, the I/O pins, the control pins, and the registers are driven to their reset states. (Table B-5 on page B-13 lists the reset states of the pins. See Table C-2 on page C-2 for the reset values of the SFRs.) The device remains in its reset state until RESET# is deasserted. When RESET# is deasserted, the bus controller fetches the chip configuration bytes (CCBs), loads them into the chip configuration registers (CCRs), and then fetches the first instruction. Figure 11-7 shows the reset-sequence timing.

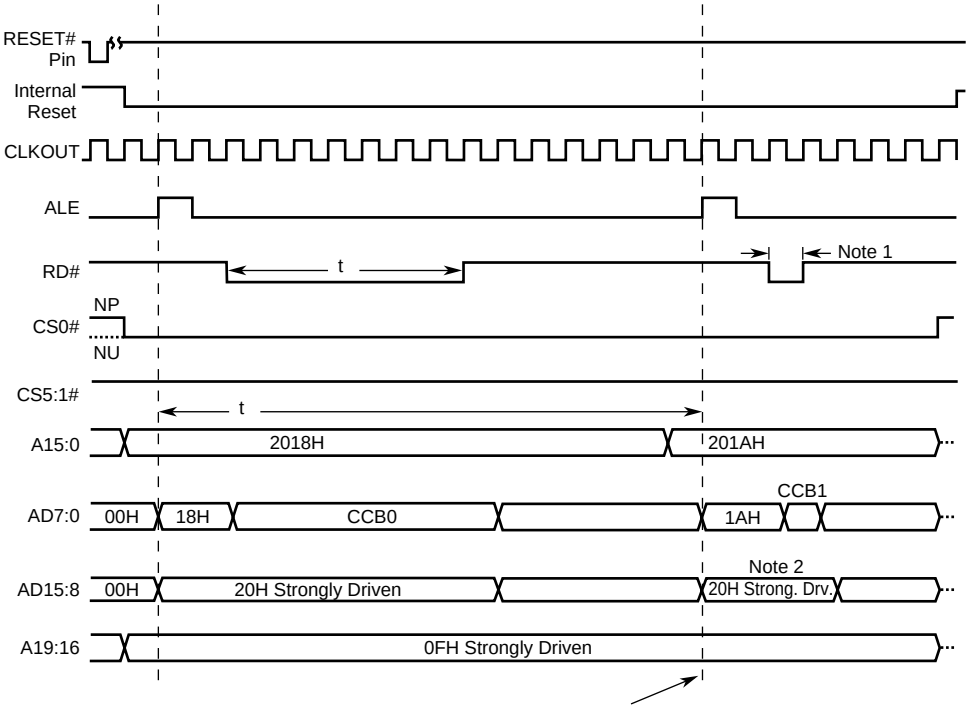


Figure 11-7. Reset Timing Sequence

The following events will reset the device (see Figure 11-8):

- an external device pulls the RESET# pin low
- the CPU issues the reset (RST) instruction
- the CPU issues an idle/powerdown (IDLPD) instruction with an illegal key operand

The following paragraphs describe each of these reset methods in more detail.

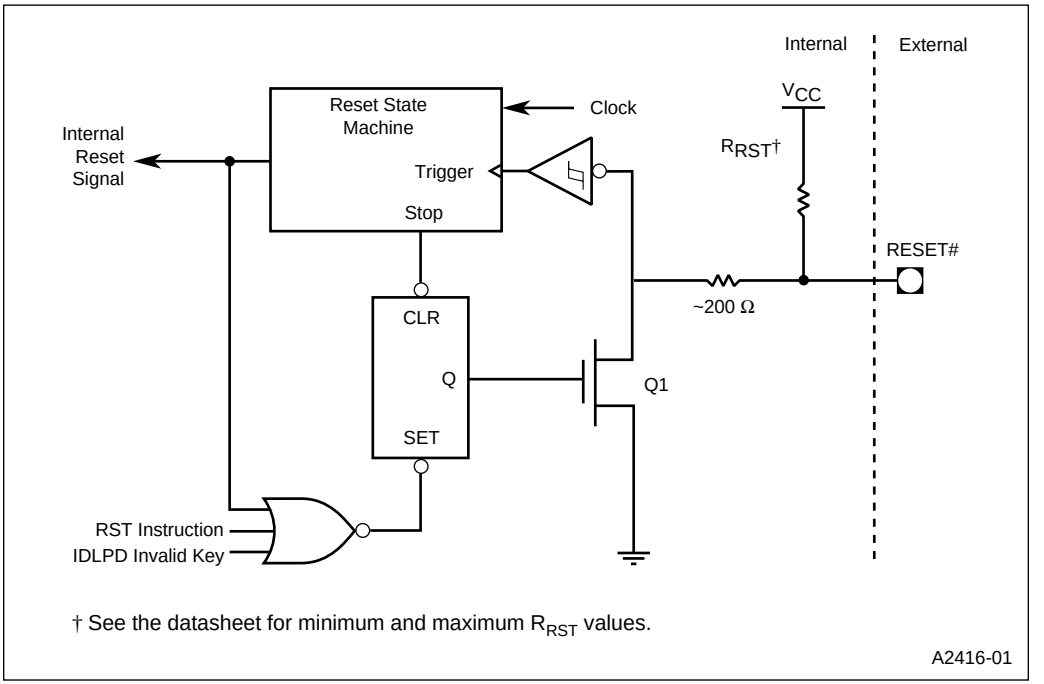


Figure 11-8. Internal Reset Circuitry

11.6.1 Generating an External Reset

To reset the device, hold the RESET# pin low for at least one state time after the power supply is within tolerance and the oscillator has stabilized. When RESET# is first asserted, the device turns on a pull-down transistor (Q1) for 16 state times. This enables the RESET# signal to function as the system reset.

The simplest way to reset the device is to insert a capacitor between the RESET# pin and V_{SS}, as shown in Figure 11-9. The device has an internal pull-up resistor (R_{RST}) shown in Figure 11-8. RESET# should remain asserted for at least one state time after V_{CC} and XTAL1 have stabilized and met the operating conditions specified in the datasheet. A capacitor of 4.7 µF or greater should provide sufficient reset time, as long as V_{CC} rises quickly.

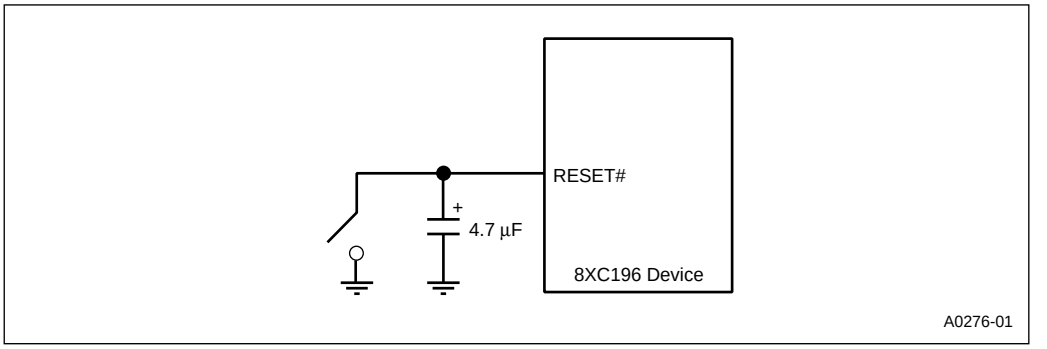


Figure 11-9. Minimum Reset Circuit

Other devices in the system may not be reset because the capacitor will keep the voltage above V_{IL}. Since RESET# is asserted for only 16 state times, it may be necessary to lengthen and buffer the system-reset pulse. Figure 11-10 shows an example of a system-reset circuit. In this example, D2 creates a wired-OR gate connection to the reset pin. An internal reset, system power-up, or SW1 closing will generate the system-reset signal.

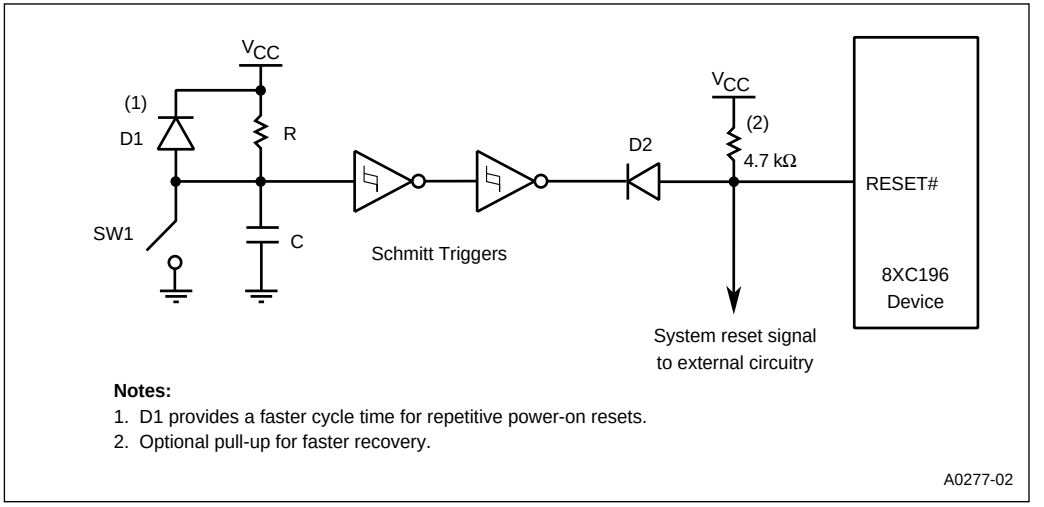


Figure 11-10. Example System Reset Circuit

11.6.2 Issuing the Reset (RST) Instruction

The RST instruction (opcode FFH) resets the device by pulling RESET# low for 16 state times. It also clears the processor status word (PSW), sets the extended and master program counters (EPC/PC) to FF2080H, and resets the special function registers (SFRs). See Table C-2 on page C-2 for the reset values of the SFRs.

11.6.3 Issuing an Illegal IDLPD Key Operand

The device resets itself if an illegal key operand is used with the idle/powerdown (IDLPD) command. The legal keys are “1” for idle mode, “2” for powerdown mode, and “3” for standby mode (NU only). If any other value is used, the device executes a reset sequence. (See Appendix A for a description of the IDLPD command.)

12

Special Operating Modes

CHAPTER 12
SPECIAL OPERATING MODES

The 8XC196NP and 80C196NU provide the following power saving modes: idle, standby (80C196NU only), and powerdown. They also provide an on-circuit emulation (ONCE) mode that electrically isolates the device from the other system components. This chapter describes each mode and explains how to enter and exit each. (Refer to Appendix A for descriptions of the instructions discussed in this chapter, to Appendix B for descriptions of signal status during each mode, and to Appendix C for details about the registers.)

12.1 SPECIAL OPERATING MODE SIGNALS AND REGISTERS

Table 12-1 lists the signals and Table 12-2 lists the registers that are mentioned in this chapter.

Table 12-1. Operating Mode Control Signals

Port Pin	Signal Name	Type	Description
P2.7	CLKOUT	O	Clock Output Output of the internal clock generator. The CLKOUT frequency is ½ the internal operating frequency (f). CLKOUT has a 50% duty cycle. CLKOUT is multiplexed with P2.7.
P3.7 P3.6 P2.4 P2.2	EXTINT3 EXTINT2 EXTINT1 EXTINT0	I	External Interrupts In normal operating mode, a rising edge on EXTINT*x* sets the EXTINT*x* interrupt pending bit. EXTINT*x* is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In standby and powerdown modes, asserting the EXTINT*x* signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled, but the pin must be configured as a special-function input (see “Bidirectional Port Pin Configurations” on page 7-7). If the EXTINT*x* interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation.
—	ONCE	I	On-circuit Emulation Holding ONCE high during the rising edge of RESET# places the device into on-circuit emulation (ONCE) mode. This mode puts all pins into a high-impedance state, thereby isolating the device from other components in the system. The value of ONCE is latched when the RESET# pin goes inactive. While the device is in ONCE mode, you can debug the system using a clip-on emulator. To exit ONCE mode, reset the device by pulling the RESET# signal low. To prevent accidental entry into ONCE mode, connect the ONCE pin to V_{SS}.

Table 12-1. Operating Mode Control Signals (Continued)

Port Pin	Signal Name	Type	Description
—	PLLEN2:1 (80C196NU only)	I	Phase Lock Loop 1 and 2 Enable These input pins are used to enable the on-chip clock multiplier feature and select either the doubled or quadrupled clock speed. **CAUTION:** If PLLEN1 is held low while PLLEN2 is held high, the device will enter into an unsupported test mode.
—	RESET#	I/O	Reset A level-sensitive reset input to and open-drain system reset output from the microcontroller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown, standby, and idle modes, asserting RESET# causes the chip to reset and return to normal operating mode. After a device reset, the first instruction fetch is from FF2080H (or F2080H in external memory). For the 80C196NP and 80C196NU, the program and special-purpose memory locations (FF2000–FF2FFFH) reside in external memory. For the 83C196NP, these locations can reside either in external memory or in internal ROM.
—	RPD	I	Return from Powerdown Timing pin for the return-from-powerdown circuit. If your application uses powerdown mode, connect a capacitor† between RPD and V_{SS} if **either** of the following conditions is true. • the internal oscillator is the clock source • the phase-locked loop (PLL) circuitry (80C196NU only) is enabled (see PLLEN2:1 signal description) The capacitor causes a delay that enables the oscillator and PLL circuitry to stabilize before the internal CPU and peripheral clocks are enabled. The capacitor is not required if your application uses powerdown mode and if **both** of the following conditions are true. • an external clock input is the clock source • the phase-locked loop circuitry (80C196NU only) is disabled If your application does not use powerdown mode, leave this pin unconnected. † Calculate the value of the capacitor using the formula found on page 12-11.

Table 12-2. Operating Mode Control and Status Registers

Mnemonic	Address	Description
CCR0	2018H	Chip Configuration 0 Register Bit 0 of this register enables and disables standby and powerdown mode.
INT_MASK	0008H	Interrupt Mask Bits 3 and 4 of this register enable and disable (mask) the external interrupts, EXTINT0 and EXTINT1.

Table 12-2. Operating Mode Control and Status Registers (Continued)

Mnemonic	Address	Description
INT_MASK1	0013H	Interrupt Mask 1 Bits 5 and 6 of this register enable and disable (mask) the external interrupts, EXTINT2 and EXTINT3.
INT_PEND	0009H	Interrupt Pending Bits 3 and 4 of this register are set to indicate a pending external interrupt, EXTINT0 and EXTINT1.
INT_PEND1	0012H	Interrupt Pending 1 Bits 5 and 6 of this register are set to indicate a pending external interrupt, EXTINT2 and EXTINT3.
P2_DIR P3_DIR	1FD3H 1FDAH	Port *x* Direction Each bit of P*x*_DIR controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as an input or open-drain output. (Open-drain outputs require external pull-ups.)
P2_MODE P3_MODE	1FD1H 1FD8H	Port *x* Mode Each bit of P*x*_MODE controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal. Setting a bit configures a pin as a special-function signal; clearing a bit configures a pin as a standard I/O port pin.
P2_REG P3_REG	1FD5H 1FDCH	Port *x* Data Output For an input, set the corresponding P*x*_REG bit. For an output, write the data to be driven out by each pin to the corresponding bit of P*x*_REG. When a pin is configured as standard I/O (P*x*_MODE.*y* = 0), the result of a CPU write to P*x*_REG is immediately visible on the pin. When a pin is configured as a special-function signal (P*x*_MODE.*y* = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to P*x*_REG, but the pin is unaffected until it is switched back to its standard I/O function. This feature allows software to configure a pin as standard I/O (clear P*x*_MODE.*y*), initialize or overwrite the pin value, then configure the pin as a special-function signal (set P*x*_MODE.*y*). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

12.2 REDUCING POWER CONSUMPTION

Each power-saving mode conserves power by disabling portions of the internal clock circuitry (Figure 12-1 and Figure 12-2). The following paragraphs describe each mode in detail.

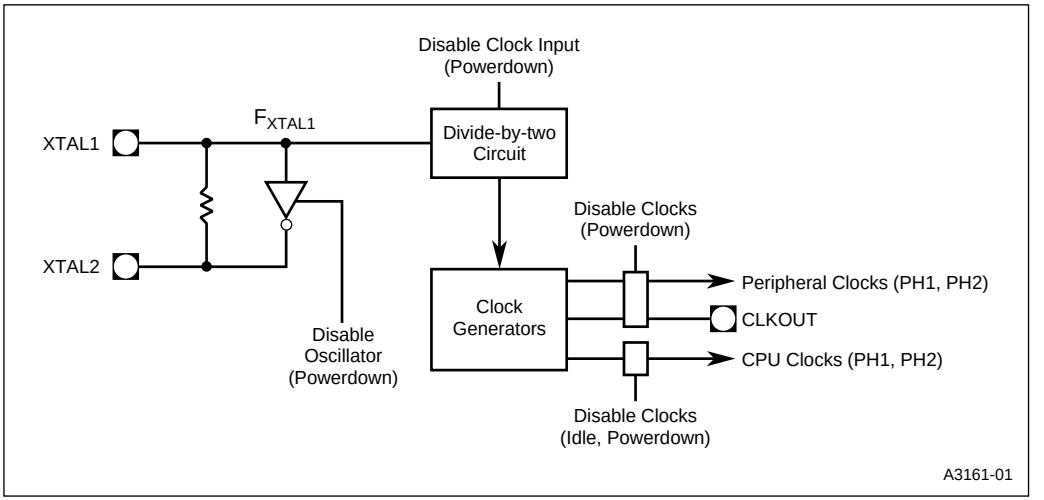


Figure 12-1. Clock Control During Power-saving Modes (8XC196NP)

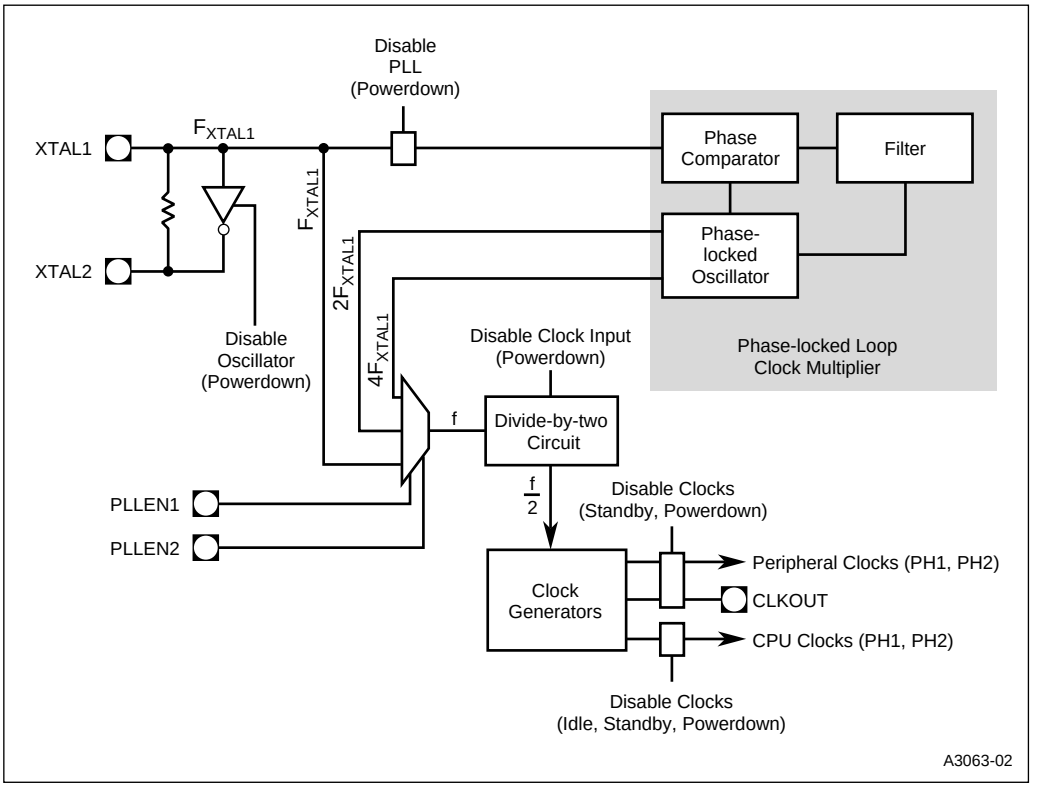


Figure 12-2. Clock Control During Power-saving Modes (80C196NU)

12.3 IDLE MODE

In idle mode, the device's power consumption decreases to approximately 40% of normal consumption. Internal logic holds the CPU clocks at logic zero, causing the CPU to stop executing instructions. Neither the phased-locked loop circuitry (80C196NU only), the peripheral clocks, nor CLKOUT are affected, so the special-function registers (SFRs) and register RAM retain their data and the peripherals and interrupt system remain active. Table B-5 on page B-13 lists the values of the pins during idle mode.

The device enters idle mode after executing the IDLPD #1 instruction. Any enabled interrupt source, either internal or external, or a hardware reset can cause the device to exit idle mode. When an interrupt occurs, the CPU clocks restart and the CPU executes the corresponding interrupt service or PTS routine. When the routine is complete, the CPU fetches and then executes the instruction that follows the IDLPD #1 instruction.

NOTE

To prevent an accidental return to full power, hold the external interrupt pins (EXTINT*x*) low while the device is in idle mode.

12.4 STANDBY MODE (80C196NU ONLY)

In standby mode, the device's power consumption decreases to approximately 10% of normal consumption. Internal logic holds the CPU and peripheral clocks at logic zero, which causes the CPU to stop executing instructions, the system bus control signals to become inactive, and the peripherals to turn off. The phase-locked loop (PLL) circuitry and the on-chip oscillator continue to operate. Table B-5 on page B-13 lists the values of the pins during standby mode.

12.4.1 Enabling and Disabling Standby Mode

Setting the PD bit in the chip-configuration register 0 (CCR0.0) enables both standby and powerdown modes. Clearing it disables both modes. CCR0 is loaded from the chip configuration byte (CCB0) when the device is reset.

12.4.2 Entering Standby Mode

Before entering standby mode, complete the following tasks:

- Complete all serial port transmissions or receptions. Otherwise, when the device exits standby, the serial port activity will continue where it left off and incorrect data may be transmitted or received.
- Put all other peripherals into an inactive state.

After completing these tasks, execute the IDLPD #3 instruction to enter standby mode.

NOTE

To prevent an accidental return to full power, hold the external interrupt pins (EXTINT*x*) low while the device is in standby mode.

12.4.3 Exiting Standby Mode

The device will exit standby mode when a transition on an **external** interrupt pin (EXTINT3:0) or a hardware reset occurs. The interrupts need not be enabled for them to bring the device out of standby, but the pin must be configured as a special-function input (see “Bidirectional Port Pin Configurations” on page 7-7).

When an external interrupt brings the device out of standby mode, the corresponding pending bit is set in the interrupt pending register. If the interrupt is enabled, the device executes the interrupt service routine, then fetches and executes the instruction following the IDLPD #3 instruction. If the interrupt is disabled (masked), the device fetches and executes the instruction following the IDLPD #3 instruction and the pending bit remains set until the interrupt is serviced or software clears it.

12.5 POWERDOWN MODE

Powerdown mode places the device into a very low power state by disabling the internal oscillator, the phase-locked loop circuitry (80C196NU only), and clock generators. Internal logic holds the CPU and peripheral clocks at logic zero, which causes the CPU to stop executing instructions, the system bus-control signals to become inactive, the CLKOUT signal to become high, and the peripherals to turn off. Power consumption drops into the microwatt range (refer to the datasheet for exact specifications). I_{CC} is reduced to device leakage. Table B-5 on page B-13 lists the values of the pins during powerdown mode. If V_{CC} is maintained above the minimum specification, the special-function registers (SFRs) and register RAM retain their data.

12.5.1 Enabling and Disabling Powerdown Mode

Setting the PD bit in the chip-configuration register 0 (CCR0.0) enables both standby and powerdown modes. Clearing it disables both modes. CCR0 is loaded from the chip configuration byte (CCB0) when the device is reset.

12.5.2 Entering Powerdown Mode

Before entering powerdown, complete the following tasks:

- Complete all serial port transmissions or receptions. Otherwise, when the device exits powerdown, the serial port activity will continue where it left off and incorrect data may be transmitted or received.
- Put all other peripherals into an inactive state.
- To allow other devices to control the bus while the microcontroller is in powerdown, assert HLDA#. Do this only if the routines for entering and exiting powerdown do not require access to external memory.

After completing these tasks, execute the IDLPD #2 instruction to enter powerdown mode.

NOTE

To prevent an accidental return to full power, hold the external interrupt pins (EXTINT*x*) low while the device is in powerdown mode.

12.5.3 Exiting Powerdown Mode

The device will exit powerdown mode when either of the following events occurs:

- a hardware reset is generated, or
- a transition occurs on an external interrupt pin.

NOTE

It was previously documented that the method of exiting powerdown mode by driving the RPD pin low was acceptable; however, we no longer recommend this method as an option for exiting powerdown.

12.5.3.1 Generating a Hardware Reset

The device will exit powerdown if RESET# is asserted. If the phase-locked loop circuitry is enabled or if the design uses an external clock input signal rather than the on-chip oscillator, RESET# must remain low for at least 16 state times. If the design uses the on-chip oscillator, then RESET# must be held low until the oscillator and phase-locked loop circuitry have stabilized.

12.5.3.2 Asserting an External Interrupt Signal

The final way to exit powerdown mode is to assert an external interrupt signal (EXTINT3:0) for at least one state time. Although EXTINT3:0 are normally sampled inputs, the powerdown circuitry uses them as level-sensitive inputs. The interrupts need not be enabled to bring the device out of powerdown, but the pin must be configured as a special-function input (see "Bidirectional Port Pin Configurations" on page 7-7). Figure 12-3 shows the power-up and powerdown sequence when using an external interrupt to exit powerdown.

When an external interrupt brings the device out of powerdown mode, the corresponding pending bit is set in the interrupt pending register. If the interrupt is enabled, the device executes the interrupt service routine, then fetches and executes the instruction following the IDLPD #2 instruction. If the interrupt is disabled (masked), the device fetches and executes the instruction following the IDLPD #2 instruction and the pending bit remains set until the interrupt is serviced or software clears the pending bit.

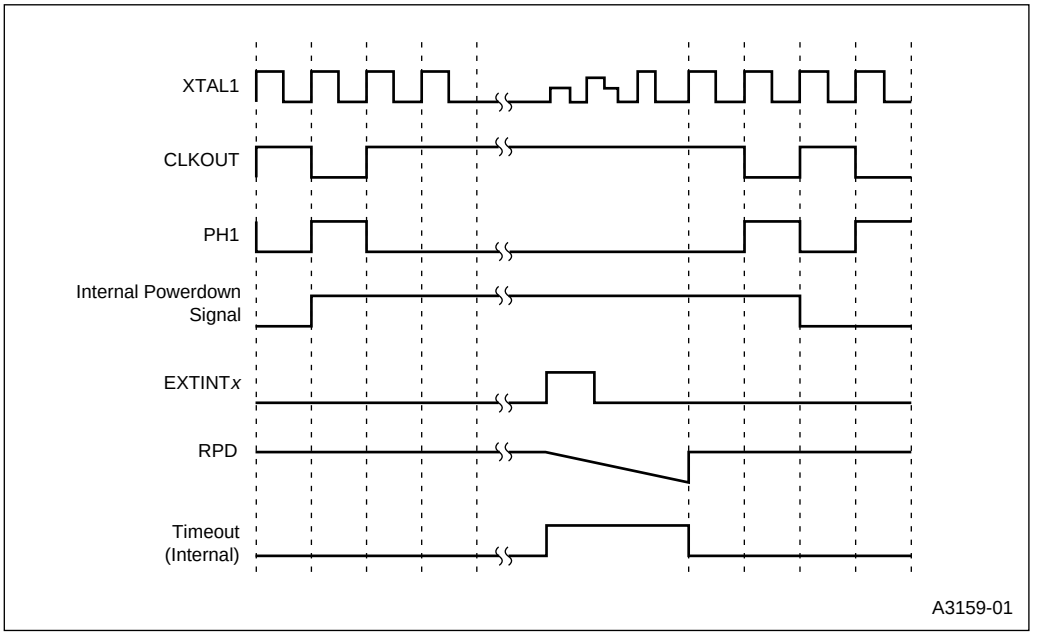


Figure 12-3. Power-up and Powerdown Sequence When Using an External Interrupt

When using an external interrupt signal to exit powerdown mode, we recommend that you connect the external component shown in Figure 12-4 to the RPD pin. The discharging of the capacitor causes a delay that allows the oscillator and phase-locked loop circuitry to stabilize before the internal CPU and peripheral clocks are enabled.

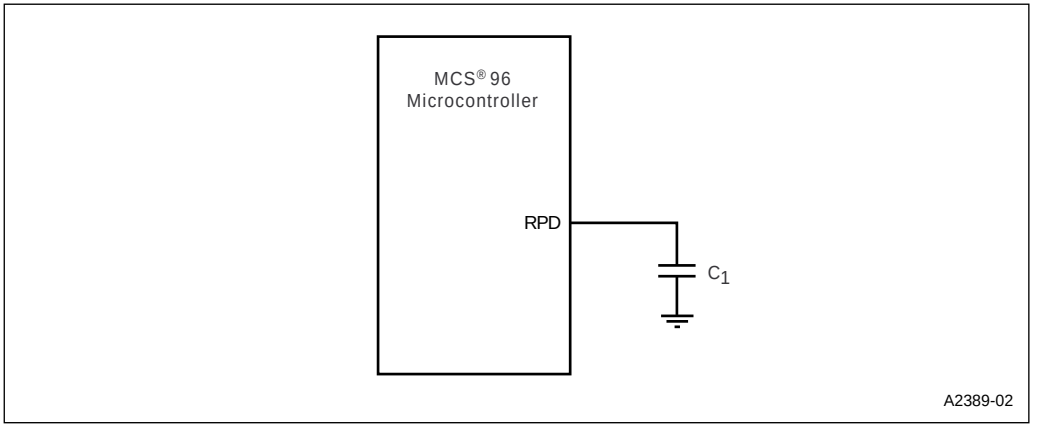


Figure 12-4. External RC Circuit

During normal operation (before entering powerdown mode), an internal pull-up holds the RPD pin at V_{CC}. When an external interrupt signal is asserted, the internal oscillator circuitry is enabled and turns on a weak internal pull-down. The resistance of the internal pull-down should be approximately 10 kΩ. This weak pull-down causes the external capacitor (C_1) to begin discharging at a typical rate of 200 µA. When the RPD pin voltage drops below the threshold voltage (about 2.5 V for 5 V operation and 1.6 V for 3 V operation), the internal phase clocks are enabled and the device resumes code execution.

At this time, a Schmitt-triggered detection circuit prompted by the switching voltage levels strongly drives a logic one, quickly pulling the RPD pin back up to V_{CC} (see recovery time in Figure 12-5). The time constant (RC) follows an exponential charging curve. However, since there is no external resistor on the RPD pin, the time constant goes to zero and the recovery time is instantaneous.

$$V_c = V_{cc}[1 - e^{(t/\tau)}]; \quad (\tau = RC_1 = 0)$$

$$V_c = V_{cc}$$

where:

V_C = Charging capacitor voltage

12.5.3.3 Selecting C_1

With the resistance of the discharge path designed into the silicon via the internal pull-down, the selection of an external capacitor (C_1) can be critical. Ideally, you want to select a component that will produce a sufficient discharge time to permit the internal oscillator circuitry to stabilize. Because many factors can influence the discharge time requirement, you should always fully characterize your design under worst-case conditions to verify proper operation.

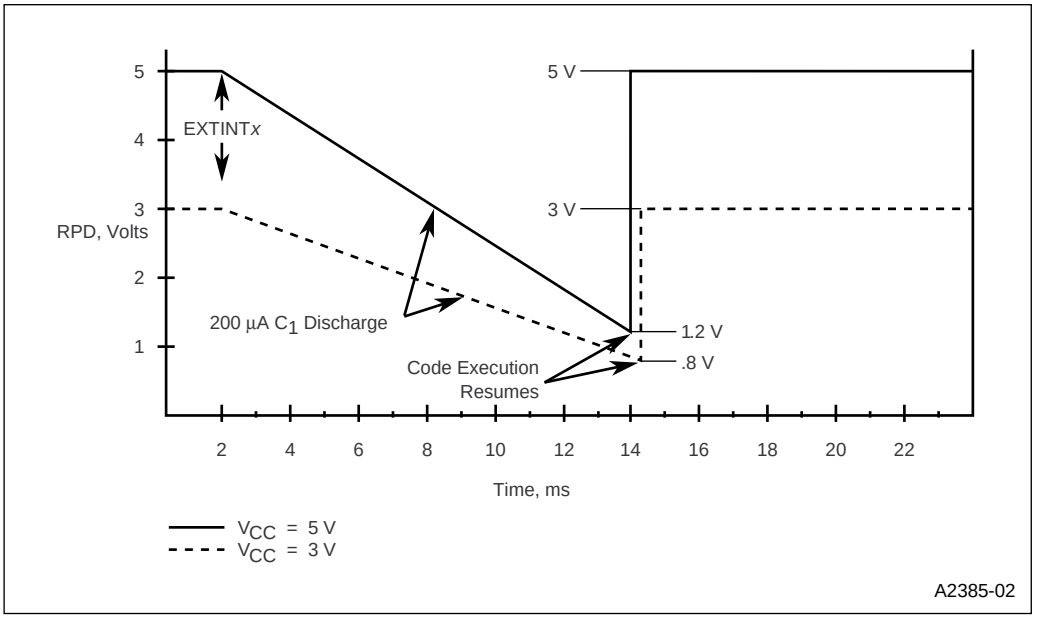


Figure 12-5. Typical Voltage on the RPD Pin While Exiting Powerdown

When selecting the capacitor, determine the worst-case discharge time needed for the oscillator to stabilize, then use this formula to calculate an appropriate value for C_1.

$$C_1 = \frac{T_{DIS} \times I}{V_t}$$

where:

C_1	is the capacitor value, in farads
T_{DIS}	is the worst-case discharge time, in seconds
I	is the discharge current, in amperes
V_t	is the threshold voltage

NOTE

If powerdown is re-entered and exited before C_1 charges to V_{CC}, it will take less time for the voltage to ramp down to the threshold. Therefore, the device will take less time to exit powerdown.

For example, assume that the oscillator needs at least 12.5 ms to discharge (T_{DIS} = 12.5 ms), V_t is 2.5 V, and the discharge current is 200 µA. The minimum C_1 capacitor size is 1 µF.

$$C_1 = \frac{(0.0125)\ (0.0002)}{2.5} = 1\ \mu F$$

When using an external oscillator, the value of C_1 can be very small, allowing rapid recovery from powerdown. For example, a 100 pF capacitor discharges in 1.25 µs.

$$T_{DIS} = \frac{C_1 \times V_t}{I} = \frac{(1.0 \times 10^{-10})\ (2.5)}{0.0002} = 1.25\ \mu s$$

12.6 ONCE MODE

On-circuit emulation (ONCE) mode isolates the device from other components in the system to allow printed-circuit-board testing or debugging with a clip-on emulator. During ONCE mode, all pins except XTAL1, XTAL2, V_{SS}, and V_{CC} are weakly pulled high or low. During ONCE mode, RESET# must be held high or the device will exit ONCE mode and enter the reset state.

Holding the ONCE signal high during the rising edge of RESET# causes the device to enter ONCE mode. The ONCE signal is latched when RESET# goes inactive. Internally, the ONCE pin is tied to a medium-strength pull-down. To prevent accidental entry into ONCE mode, connect the ONCE pin to V_{SS}.

Exit ONCE mode by asserting the RESET# signal. Normal operations resume when RESET# goes high.

12.7 RESERVED TEST MODES (80C196NU ONLY)

For the 80C196NU only, holding PLLEN1 low while PLLEN2 is held high causes the device to enter an unsupported test mode. Table 12-3 shows the proper PLLEN1 and PLLEN2 connections for valid clock modes.

Table 12-3. 80C196NU Clock Modes

PLLEN2	PLLEN1	Mode
0	0	Clock-multiplier circuitry disabled.
0	1	Reserved. **CAUTION:** This combination causes the device to enter an unsupported test mode.
1	0	Doubled; clock doubling circuitry enabled. Internal clock is twice the XTAL1 input.
1	1	Quadrupled; clock quadrupling circuitry enabled. Internal clock is four times the XTAL1 input.

13

Interfacing with External Memory

CHAPTER 13
INTERFACING WITH EXTERNAL MEMORY

The device can interface with a variety of external memory devices. Six chip-selects can be individually programmed for bus width, the number of wait states, and a multiplexed or demultiplexed address/data bus. Other features of the external memory interface include ready control for inserting additional wait states, a bus-hold protocol that enables external devices to take control of the bus, and two write-control modes for writing words and bytes to memory. These features provide a great deal of flexibility when interfacing with external memory devices.

In addition to describing the signals and registers related to external memory, this chapter discusses the process of fetching the chip configuration bytes and configuring the external bus. It also provides examples of external memory configurations and chip-select setup.

13.1 INTERNAL AND EXTERNAL ADDRESSES

The address that external devices see is different from the address that the device generates internally. Internally, the device has 24 address lines, but only the lower 20 address lines (A19:0) are implemented with external pins. The absence of the upper four address bits at the external pins causes different internal addresses to have the same external address. For example, the internal addresses FF2080H, 7F2080H, and 0F2080H all appear at the 20 external pins as F2080H. The upper nibble of the internal address has no effect on the external address.

The address seen by an external device also depends on the number of address lines that the external system uses. If the address on the external pins (A19:0) is F2080H, and only A17:0 are connected to the external device, the external device sees 32080H. The upper four address lines (A19:16) are implemented by the EPORT. Table 13-1 shows how the external address depends on the number of EPORT lines used to address the external device.

Table 13-1. Example of Internal and External Addresses

EPORT Lines Connected to the External Device	Internal Address	Address on the Device Pins	Address Seen by External Device
A16	*x*F2080H	F2080H	12080H
A17:16	*x*F2080H	F2080H	32080H
A18:16	*x*F2080H	F2080H	72080H
A19:16	*x*F2080H	F2080H	F2080H

13.2 EXTERNAL MEMORY INTERFACE SIGNALS

Table 13-2 describes the external memory interface signals. For some signals, the pin has an alternate function (shown in the *Multiplexed With* column). In some cases the alternate function is a port signal (e.g., P2.7). Chapter 7, "I/O Ports," describes how to configure a pin for its I/O port function and for its special function. In other cases, the signal description includes instructions for selecting the alternate function.

Table 13-2. External Memory Interface Signals

Name	Type	Description	Multiplexed With
A15:0	I/O	System Address Bus These address lines provide address bits 15–0 during the entire external memory cycle during both multiplexed and demultiplexed bus modes.	—
A19:16	I/O	Address Lines 16–19 These address lines provide address bits 16–19 during the entire external memory cycle, supporting extended addressing of the 1 Mbyte address space. **NOTE:** Internally, there are 24 address bits; however, only 20 address lines (A19:0) are bonded out. The internal address space is 16 Mbytes (000000–FFFFFFH) and the external address space is 1 Mbyte (00000–FFFFFH). The device resets to FF2080H in internal ROM or F2080H in external memory.	EPORT.3:0
AD15:0	I/O	Address/Data Lines The function of these pins depend on the bus size and mode. When a bus access is not occurring, these pins revert to their I/O port function. **16-bit Multiplexed Bus Mode**: AD15:0 drive address bits 0–15 during the first half of the bus cycle and drive or receive data during the second half of the bus cycle. **8-bit Multiplexed Bus Mode**: AD15:8 drive address bits 8–15 during the entire bus cycle. AD7:0 drive address bits 0–7 during the first half of the bus cycle and either drive or receive data during the second half of the bus cycle. **16-bit Demultiplexed Mode**: AD15:0 drive or receive data during the entire bus cycle. **8-bit Demultiplexed Mode**: AD7:0 drive or receive data during the entire bus cycle. AD15:8 drive the data that is currently on the high byte of the internal bus.	—

Table 13-2. External Memory Interface Signals (Continued)

Name	Type	Description	Multiplexed With
ALE	O	Address Latch Enable This active-high output signal is asserted only during external memory cycles. ALE signals the start of an external bus cycle and indicates that valid address information is available on the system address/data bus (A19:16 and AD15:0 for a multiplexed bus; A19:0 for a demultiplexed bus). ALE differs from ADV# in that it does not remain active during the entire bus cycle. An external latch can use this signal to demultiplex address bits 0–15 from the address/data bus in multiplexed mode.	—
BHE#	O	Byte High Enable† During 16-bit bus cycles, this active-low output signal is asserted for word reads and writes and high-byte reads and writes to external memory. BHE# indicates that valid data is being transferred over the upper half of the system data bus. Use BHE#, in conjunction with A0, to determine which memory byte is being transferred over the system bus: **BHE# A0 Byte(s) Accessed** 0 0 both bytes 0 1 high byte only 1 0 low byte only † The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.	P5.5/WRH#
BREQ#	O	Bus Request This active-low output signal is asserted during a hold cycle when the bus controller has a pending external memory cycle. The device can assert BREQ# at the same time as or after it asserts HLDA#. Once it is asserted, BREQ# remains asserted until HOLD# is removed. You must enable the bus-hold protocol before using this signal (see “Enabling the Bus-hold Protocol” on page 13-32).	P2.3
CLKOUT	O	Clock Output Output of the internal clock generator. The CLKOUT frequency is ½ the internal operating frequency (f). CLKOUT has a 50% duty cycle.	P2.7
CS5:0#	O	Chip-select Lines 0–5 The active-low output CS*x*# is asserted during an external memory cycle when the address to be accessed is in the range programmed for chip select *x*. If the external memory address is outside the range assigned to the six chip selects, no chip-select output is asserted and the bus configuration defaults to the CS5# values. Immediately following reset, CS0# is automatically assigned to the range FF2000–FF20FFH (F2000–F20FFH if external).	P3.5:0

Table 13-2. External Memory Interface Signals (Continued)

Name	Type	Description	Multiplexed With
EA#	I	External Access This input determines whether memory accesses to special-purpose and program memory partitions (FF2000–FF2FFFH) are directed to internal or external memory. These accesses are directed to internal memory if EA# is held high and to external memory if EA# is held low. For an access to any other memory location, the value of EA# is irrelevant. EA# is not latched and can be switched dynamically during normal operating mode. Be sure to thoroughly consider the issues, such as different access times for internal and external memory, before using this dynamic switching capability. On devices with no internal nonvolatile memory, always connect EA# to V_{SS}. EA# is not implemented on the 80C196NU.	—
HLDA#	O	Bus Hold Acknowledge This active-low output indicates that the CPU has released the bus as the result of an external device asserting HOLD#.	P2.6
HOLD#	I	Bus Hold Request An external device uses this active-low input signal to request control of the bus. This pin functions as HOLD# only if the pin is configured for its special function (see "Bidirectional Port Pin Configurations" on page 7-7) and the bus-hold protocol is enabled. Setting bit 7 of the window selection register (WSR) enables the bus-hold protocol.	P2.5
INST	O	Instruction Fetch This active-high output signal is valid only during external memory bus cycles. When high, INST indicates that an instruction is being fetched from external memory. The signal remains high during the entire bus cycle of an external instruction fetch. INST is low for data accesses, including interrupt vector fetches and chip configuration byte reads. INST is low during internal memory fetches.	—
RD#	O	Read Read-signal output to external memory. RD# is asserted only during external memory reads.	—
READY	I	Ready Input This active-high input signal is used to lengthen external memory cycles for slow memory by generating wait states in addition to the wait states that are generated internally. When READY is high, CPU operation continues in a normal manner with wait states inserted as programmed in CCR0 or the chip-select *x* bus control register. READY is ignored for all internal memory accesses.	—

Table 13-2. External Memory Interface Signals (Continued)

Name	Type	Description	Multiplexed With
WR#	O	Write† This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes. † The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.	WRL#
WRH#	O	Write High† During 16-bit bus cycles, this active-low output signal is asserted for high-byte writes and word writes to external memory. During 8-bit bus cycles, WRH# is asserted for all write operations. † The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.	P5.5/BHE#
WRL#	O	Write Low† During 16-bit bus cycles, this active-low output signal is asserted for low-byte writes and word writes. During 8-bit bus cycles, WRL# is asserted for all write operations. † The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.	WR#

13.3 THE CHIP-SELECT UNIT

The chip-select unit provides six outputs, CS5:0#, for selecting an external device during an external bus cycle. During an external memory access, a chip-select output CS*x*# is asserted if the address falls within the address range assigned to that chip-select. The bus width, the number of wait states, and multiplexed or demultiplexed address/data lines are programmed independently for each of the six chip-selects. If the external address is outside the range of the six chip-selects, the chip-select 5 bus control register determines the wait states, bus width, and multiplexing, and no chip-select is asserted. Table 13-3 lists the chip-select registers.

Table 13-3. Chip-select Registers

Register Mnemonic	Address	Description
ADDRCOM0 ADDRCOM1 ADDRCOM2 ADDRCOM3 ADDRCOM4 ADDRCOM5	1F40H 1F48H 1F50H 1F58H 1F60H 1F68H	Address Compare Register This 16-bit register holds the upper 12 bits of the base address of the address range assigned to CS*x*#.
ADDRMSK0 ADDRMSK1 ADDRMSK2 ADDRMSK3 ADDRMSK4 ADDRMSK5	1F42H 1F4AH 1F52H 1F5AH 1F62H 1F6AH	Address Mask Register This register determines the size of the address range (256 bytes–1 Mbyte).
BUSCON0 BUSCON1 BUSCON2 BUSCON3 BUSCON4 BUSCON5	1F44H 1F4CH 1F54H 1F5CH 1F64H 1F6CH	Bus Control Register This register determines the bus configuration for external accesses to the address range assigned to CS*x*#. The bus parameters are 8- or 16-bit bus width, multiplexed or demultiplexed address/data lines, and the number of wait states inserted into each bus cycle.

Figure 13-1 illustrates the device's calculation of a chip-select output CS*x*# for a given external memory address. The 12 most-significant bits of the external address are compared (XORed) bit-wise with the 12 least-significant bits (BASE19:8) of the ADDRCOM*x* register. If all of the bits match, CS*x*# is asserted. Additionally, if some bits do not match, CS*x*# is still asserted if, for each non-matching bit in ADDRCOM*x*, the corresponding bit in ADDRMSK*x* is cleared. The 12 least-significant bits are named MASK19:8 for their function in masking bits BASE19:8.

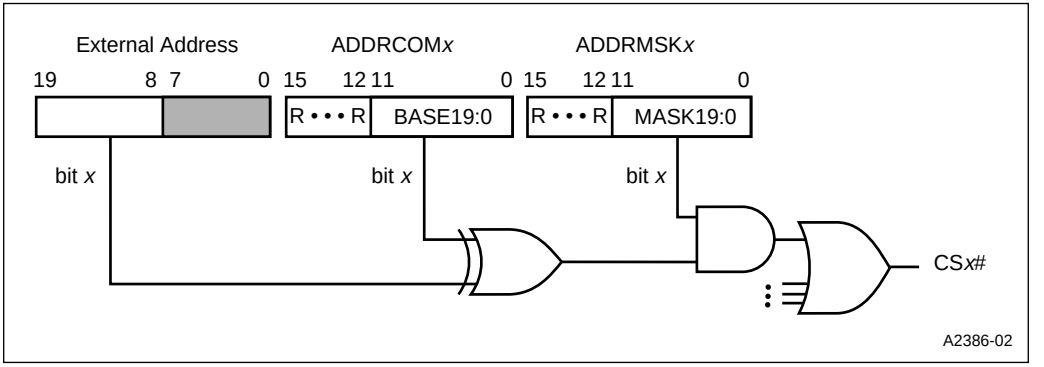


Figure 13-1. Calculation of a Chip-select Output

13.3.1 Defining Chip-select Address Ranges

This section describes the ADDRCOM*x* and ADDRMSK*x* registers and how to set them up for a desired address range. The ADDRCOM*x* register (Figure 13-2) and ADDRMSK*x* register (Figure 13-3) control the assertion of each chip-select output CS*x*#. The BASE19:8 bits in the ADDRCOM*x* register determine the base address of the address range. The MASK19:8 bits in the ADDRMSK*x* register determine the size of the address range.

ADDRCOM*x*
***x* = 0–5**

Address: Table 13-4
Reset State:

The address compare (ADDRCOM*x*) register specifies the base (lowest) address of the address range. The base address of a 2^n-byte address range must be on a 2^n-byte boundary.

15							8
—	—	—	—	BASE19	BASE18	BASE17	BASE16
7							**0**
BASE15	BASE14	BASE13	BASE12	BASE11	BASE10	BASE9	BASE8

Bit Number	Bit Mnemonic	Function
15:12	—	Reserved; for compatibility with future devices, write zeros to these bits.
11:0	BASE19:8	Base Address Bits These bits are the 12 most-significant bits of the base address of the address range assigned to chip-select *x*.

Figure 13-2. Address Compare (ADDRCOM*x*) Register

Table 13-4. ADDRCOM*x* Addresses and Reset Values

Register	Address	Reset Value
ADDRCOM0	1F40H	0F20H
ADDRCOM1	1F48H	X000H
ADDRCOM2	1F50H	X000H
ADDRCOM3	1F58H	X000H
ADDRCOM4	1F60H	X000H
ADDRCOM5	1F68H	X000H

ADDRMSK*x*
***x* = 0–5**

Address: Table 13-5
Reset State:

The address mask (ADDRMSK*x*) register, together with the address compare register, defines the address range that is assigned to the chip-select *x* output, CS*x*#. The address mask register determines the size of the address range, which must be 2^n bytes, where n = 8, 9, . . , 20. For a 2^n-byte address range, calculate $n_1 = 20 - n$, and set the n_1 most-significant bits of MASK19:8 in the address mask register.

15							8
—	—	—	—	MASK19	MASK18	MASK17	MASK16
7							**0**
MASK15	MASK14	MASK13	MASK12	MASK11	MASK10	MASK9	MASK8

Bit Number	Bit Mnemonic	Function
15:12	—	Reserved; for compatibility with future devices, write zeros to these bits.
11:0	MASK19:8	Address Mask Bits For a 2^n-byte address range, set the n_1 most-significant bits of MASK19:8, where $n_1 = 20 - n$.

Figure 13-3. Address Mask (ADDRMSK*x*) Register

Table 13-5. ADDRMSK*x* Addresses and Reset Values

Register	Address	Reset Value
ADDRMSK0	1F42H	XFFFH
ADDRMSK1	1F4AH	XFFFH
ADDRMSK2	1F52H	XFFFH
ADDRMSK3	1F5AH	XFFFH
ADDRMSK4	1F62H	XFFFH
ADDRMSK5	1F6AH	XFFFH

Observe the following restrictions in choosing an address range for a chip-select output:

- The addresses in the address range must be contiguous.
- The size of the address range must be 2^n bytes, where n = 8, 9, ..., 20. This corresponds to block sizes of 256 bytes, 512 bytes, ..., 1 Mbyte.
- The base address of a 2^n-byte address range must be on a 2^n-byte boundary (that is, the base address must be evenly divisible by 2^n). For example, the base address of a 256-Kbyte range must be 00000H, 40000H, 80000H, or C0000H. Table 13-6 shows the base addresses for some address-range sizes.
- The address ranges for different chip-selects must not overlap, unless their BUSCON*x* parameters (wait states, bus width, and multiplexing) have the same values. If BUSCON*x* registers have different parameter values and an address in their overlapping region is accessed, the results are unpredictable. See "Example of a Chip-select Setup" on page 13-12 for a chip-select initialization procedure that avoids this difficulty.

Table 13-6. Base Addresses for Several Sizes of the Address Range

Address-Range Size	1 Mbyte	512 Kbyte	256 Kbyte		512 bytes	256 bytes
Base Addresses	00000H	00000H	00000H		00000H	00000H
		80000H	40000H		00200H	00100H
			80000H	• • •	00400H	00200H
			C0000H		00600H	00300H
					• • •	• • •
					FFB00H	FFE00H
					FFD00H	FFF00H

For an address range satisfying these restrictions, set up the ADDRCOM*x* and ADDRMSK*x* registers as follows:

- Place the 12 most-significant bits of the base address into bits BASE19:8 in the ADDRCOM*x* register (Figure 13-2).
- For an address range of 2^n bytes, set the n_1 most-significant bits of MASK19:8 in the ADDRMSK*x* register (Figure 13-3), where $n_1 = 20 - n$.

For example, assume that chip-select output *x* is to be assigned to a 32-Kbyte address range with base address E0000H. The address range size is $32 \times 1024 = 2^{15}$, and $n_1 = 20 - 15 = 5$. To set up the registers, write the 12 most-significant bits of E0000H to BASE19:8 in the ADDRCOM*x* register, and set the 5 most-significant bits of MASK19:8 in the ADDRMSK*x* register:

ADDRCOM*x* = 0E00H
ADDRMSK*x* = 0F80H

Note that the 32-Kbyte address range could not have 4000H as base address, for example, because 4000H is not on a 32-Kbyte boundary.

"Example of a Chip-select Setup" on page 13-12 shows another example of setting up the chip-select unit.

13.3.2 Controlling Wait States, Bus Width, and Bus Multiplexing

For each chip-select output address range, the bus control register BUSCON*x* (Figure 13-4) determines the wait states, the bus width, and the address/data multiplexing.

BUSCON*x*
***x* = 0–5**

Address: Table 13-7
Reset State:

For the address range assigned to chip-select *x*, the bus control (BUSCON*x*) register specifies the number of wait states, the bus width, and the address/data multiplexing for all external bus cycles that access address range *x*.

7							0
DEMUX	BW16	—	—	—	—	WS1	WS0

Bit Number	Bit Mnemonic	Function
7	DEMUX	Address/Data Multiplexing This bit specifies the address/data multiplexing on AD15:0 for all external accesses to the address range assigned to chip-select *x* output. 0 = multiplexed 1 = demultiplexed
6	BW16	Bus Width This bit specifies the bus width for all external accesses to the address range assigned to chip-select *x* output. 0 = 8 bits 1 = 16 bits
5:2	—	Reserved; for compatibility with future devices, write zeros to these bits.
1:0	WS1:0	Wait States These bits specify the number of wait states for all external accesses to the address range assigned to chip-select *x* output. **WS1 WS0 Wait States** 0 0 0 0 1 1 1 0 2 1 1 3

Figure 13-4. Bus Control (BUSCON*x*) Register

Table 13-7. BUSCON*x* Addresses and Reset Values

Register	Address	Reset Value
BUSCON0	1F44H	03H
BUSCON1	1F4CH	00H
BUSCON2	1F54H	00H
BUSCON3	1F5CH	00H
BUSCON4	1F64H	00H
BUSCON5	1F6CH	00H

13.3.3 Chip-select Unit Initial Conditions

A chip reset produces the following initial conditions for the chip-select unit:

- ADDRMSK*x* = XFFFH.
- ADDRCOM0 = 0F20H. This asserts CS0# for the 256-byte address range F2000–F20FFH.
- ADDRCOM1–ADDRCOM5 = X000H.
- For the fetch of chip configuration byte 0 (CCB0), BUSCON0 is initialized for an 8-bit bus width, multiplexed mode, and three wait states (DEMUX = 0, BW16 = 0, WS0 = 1, WS1 = 1).
- Before the fetch of chip configuration byte 1 (CCB1), the values of DEMUX, BW16, WS0, and WS1 in BUSCON0 are loaded from CCB0. The external bus is configured according to the new values.

The first lines of your program should perform two tasks:

1. Set the stack pointer.
2. Initialize all of the chip-select registers (ADDRCOM*x*, ADDRMSK*x*, and BUSCON*x*, by using the procedure in “Initializing the Chip-select Registers.”

13.3.4 Initializing the Chip-select Registers

When initializing the chip-select parameters (or modifying them at any time), it is important to avoid a condition in which two chip-selects outputs have overlapping address ranges and different bus-parameter values (wait states, bus width, and multiplexing). Accessing a location in such an overlapping address range can cause unpredictable results.

Use the following sequence to initialize the chip-select registers after reset:

1. Initialize chip-select output 0:
   1.1. Clear ADDRMSK0.
   1.2. Write to ADDRCOM0 to establish the desired base address.
   1.3. Write to ADDRMSK0 to establish the desired address range.
   1.4. Write the desired bus-parameter values to BUSCON0.
2. While executing in the address range defined in step 1 for chip-select output 0, use the following sequence to initialize chip-select outputs 1–5. Begin with $x = 1$.
   2.1. Load ADDRMSK*x* with 0FFFH.
   2.2. Write to ADDRCOM*x* to establish the desired base address.
   2.3. Write to ADDRMSK*x* to establish the desired address range.
   2.4. Write the desired bus-parameter values to BUSCON*x*.
   2.5. Repeat steps 2.1–2.4 for $x = 2–5$.

13.3.5 Example of a Chip-select Setup

This section shows an example of setting up the chip-select unit and provides details of the chip-select output calculation. This example shows how to set up the chip-select registers for the system shown in Figure 13-5. For each address range, the BUSCON*x* register (see Figure 13-4) specifies the address/data multiplexing (bit 7), the bus width (bit 6), and the number of wait states (bits 1, 0). Table 13-8 lists the characteristics of the three chip-select outputs and the corresponding contents of BUSCON*x*.

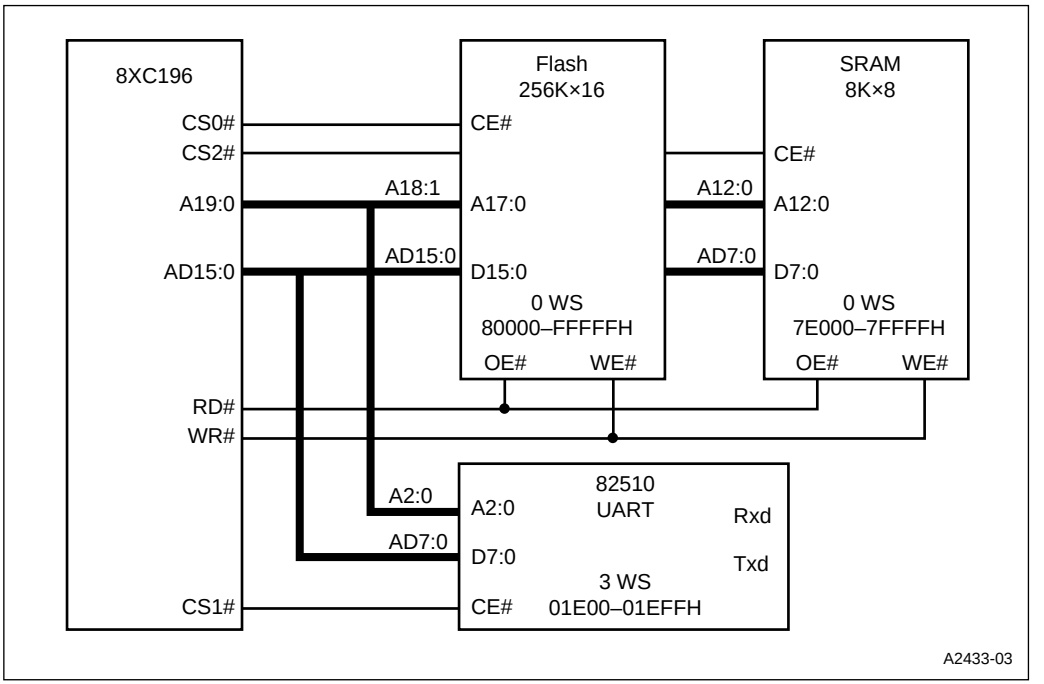


Figure 13-5. Example System for Setting Up Chip-select Outputs

Table 13-8. BUSCON*x* Registers for the Example System

Chip-select Output	Multiplexing	Bus Width	Wait States	Contents of BUSCON*x*
0	Demultiplexed	16 bits	0	C0H
1	Demultiplexed	8 bits	3	83H
2	Demultiplexed	8 bits	0	80H

The location and size of an address range are specified by the ADDRCOM*x* register and the ADDRMSK*x* register (see Figure 13-2 and Figure 13-3). The 8-Kbyte SRAM is assigned to address range 7E000–7FFFFH and uses chip-select output 2. The 12 most-significant bits of the base address (7E000H) are written to the BASE19:8 bits in the ADDRCOM2 register, which then contains 07E0H.

The address range for CS2# is 8 Kbytes or 2^{13} bytes (n = 13). The number of bits to be set in MASK19:8 of ADDRMSK2 is $20 - n = 7$. After the 7 most-significant bits of MASK19:8 are set, ADDRMSK2 contains 0FE0H. Results for CS0# and CS1# are found similarly (see Table 13-9).

Table 13-9. Results for the Chip-select Example

Chip Select	Address Range	Size of Address Range	Number of Bits to Set in ADDRMSK*x*	Contents of ADDRCOM*x*	Contents of ADDRMSK*x*
0	80000–FFFFFH	512 Kbytes = 2^{19} bytes	n_1 = 20 – 19 = 1	0800H	0800H
1	01E00–01EFFH	256 bytes = 2^8 bytes	n_1 = 20 – 8 = 12	001EH	0FFFH
2	7E000–7FFFFH	8 Kbytes = 2^{13} bytes	n_1 = 20 – 13 = 7	07E0H	0FE0H

13.4 CHIP CONFIGURATION REGISTERS AND CHIP CONFIGURATION BYTES

Two chip configuration registers (CCRs) have bits that set parameters for chip operation and external bus cycles. The CCRs cannot be accessed by code. They are loaded from the chip configuration bytes (CCBs), which have internal addresses FF2018H (CCB0) and FF201AH (CCB1). If the CCBs are stored in external memory, their external addresses depend on the number of EPORT lines used in the external system (see "Internal and External Addresses" on page 13-1).

When the device returns from reset, the bus controller fetches the CCBs and loads them into the CCRs. From this point, these CCR bit values define the chip configuration until the device is reset again. The CCR bits are described in Figures 13-6 and 13-7. The remainder of this section describes the state of the chip following reset and the process of fetching the CCBs.

CCR0 no direct access†

The chip configuration 0 (CCR0) register enables or disables powerdown and standby (80C196NU only) modes and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
1	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7:6	1	To guarantee device operation, write ones to these bits.
5:4	WS1:0	Wait States These two bits control the number of wait states that are used for an external fetch of CCB1. **WS0 WS1** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus
0	PD	Powerdown Enable Enables or disables the IDLPD #2 and IDLPD #3 instructions. When enabled, the IDLPD #2 instruction causes the microcontroller to enter powerdown mode and for the 80C196NU only, the IDLPD #3 instruction causes the microcontroller to enter standby mode. 0 = disable powerdown and standby modes 1 = enable powerdown and standby modes If your design uses powerdown or standby mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into powerdown and standby mode†. (Chapter 12, “Special Operating Modes,” discusses powerdown and standby modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

Figure 13-6. Chip Configuration 0 (CCR0) Register

CCR1 no direct access†

The chip configuration 1 (CCR1) register selects the 16-bit or 24-bit addressing mode and (for the 8XC196NP only) controls whether the internal ROM is mapped into two address ranges, FF2000–FF2FFFH and 002000–002FFFH, or into FF2000–FF2FFFH only.

	7							0
8XC196NP	1	1	0	1	1	REMAP	MODE64	—
	7							**0**
80C196NU	1	1	DM	1	1	—	MODE64	—

Bit Number	Bit Mnemonic	Function
7:6	1	To guarantee device operation, write ones to these bits.
5††	DM	Deferred Mode Enables the deferred bus-cycle mode. If the 80C196NU is using a demultiplexed bus and deferred mode is enabled, a delay of 2t occurs in the first bus cycle following a chip-select output change and the first write cycle following a read cycle. (See "Deferred Bus-cycle Mode (80C196NU Only)" on page 13-40.) 0 = deferred bus-cycle mode disabled 1 = deferred bus-cycle mode enabled
4:3	1	To guarantee device operation, write ones to these bits.
2††	REMAP	Internal ROM Mapping Controls the internal ROM mapping. 0 = ROM maps to FF2000–FF2FFFH only 1 = ROM maps to FF2000–FF2FFFH and 002000–002FFFH
1	MODE64	Addressing Mode Selects 64-Kbyte or 1-Mbyte addressing. 0 = selects 1-Mbyte addressing 1 = selects 64-Kbyte addressing
0	—	Reserved; for compatibility with future devices, write zero to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Bit 5 is reserved on the 8XC196NP device and bit 2 is reserved on the 80C196NU device. For compatibility with future devices, write zeros to these bits.

Figure 13-7. Chip Configuration 1 (CCR1) Register

Upon leaving the reset state, the device is configured for normal operation. This section describes the state of the chip following reset and summarizes the steps in the configuration process.

Following reset, the chip automatically fetches the two chip configuration bytes.

- **83C196NP only.** The CCB fetches are from external memory if EA# = 0 and from internal ROM if EA# = 1.
- **80C196NP and 80C196NU only.** The CCB fetches are from external memory. (EA# should be tied low.)

If the CCBs are stored in external ROM, chip-select output 0 (CS0#) should be connected to that device. Chip-select output 0 is initialized for the address range FF2000–FF20FFH, which includes the CCB locations. Following the CCB fetches, the device fetches the instruction at FF2080H.

The device uses the following bus control parameters for the CCB0 fetch:

- Bus multiplexing (DEMUX): multiplexed
- Bus width (BW16): 8 bits
- Wait states (WS0, WS1): 3 wait states. The READY pin is active for the CCB0 and CCB1 fetches and can be used to insert additional wait states (see “Wait States (Ready Control)” on page 13-26).

CCB0 can be fetched over a 16-bit bus, even though BW16 defaults to 8 bits for the CCB0 fetch. The upper address lines A19:8 and AD15:8 are strongly driven during the CCB0 fetch because an 8-bit bus is assumed. Therefore, if you have a 16-bit data bus, write the value 20H to FF2019H to avoid contention on AD15:8. Lines A19:0 are driven in the multiplexed mode. You can access the memory using A19:0 and use AD15:0 for data only.

CCB0 itself contains bits that specify DEMUX, BW16, WS0, and WS1. These values are used to control the CCB1 fetch, and following the fetch, they are stored in the chip-select output 0 bus control register, BUSCON0 (see “Chip-select Unit Initial Conditions” on page 13-11). The bits in CCB0 and CCB1 are described in “Chip Configuration Registers and Chip Configuration Bytes” on page 13-14.

After RESET# is deasserted, the following pins are initialized:

- The P2.7/CLKOUT pin operates as CLKOUT (as during reset). Be sure that the CLKOUT signal does not damage external hardware.
- The P3.0/CS0# pin operates as CS0#, which is asserted for the CCB fetches. If you plan to use the P3.0 pin as an input, it must be reconfigured from its post-reset operation as an output.
- The BHE#/WRH# pin operates as BHE#.
- The WR#/WRL# pin operates as WR#.
- Bus-hold function is disabled internally (WSR.7 = 0).
- The READY/P5.6 pin is active (that is, the chip responds to external requests for additional wait states).
- The INST pin is low (deasserted).
- The AD15:0 pins are active.
- The following port pins are weakly held high: P1.7:0, P2.6, P2.4:0, P3.7:1, and P4.7:0.
- The EPORT.3:0 pins are forced high, regardless of the state of the EA# pin.

Following reset, you should set the stack pointer and initialize the chip-select outputs using the procedure in "Example of a Chip-select Setup" on page 13-12.

13.5 BUS WIDTH AND MULTIPLEXING

The external bus can operate with a 16-bit or 8-bit data bus and with a multiplexed or demultiplexed address/data bus. Figure 13-8 shows the external bus signals during operation in the four combinations of bus width and multiplexing.

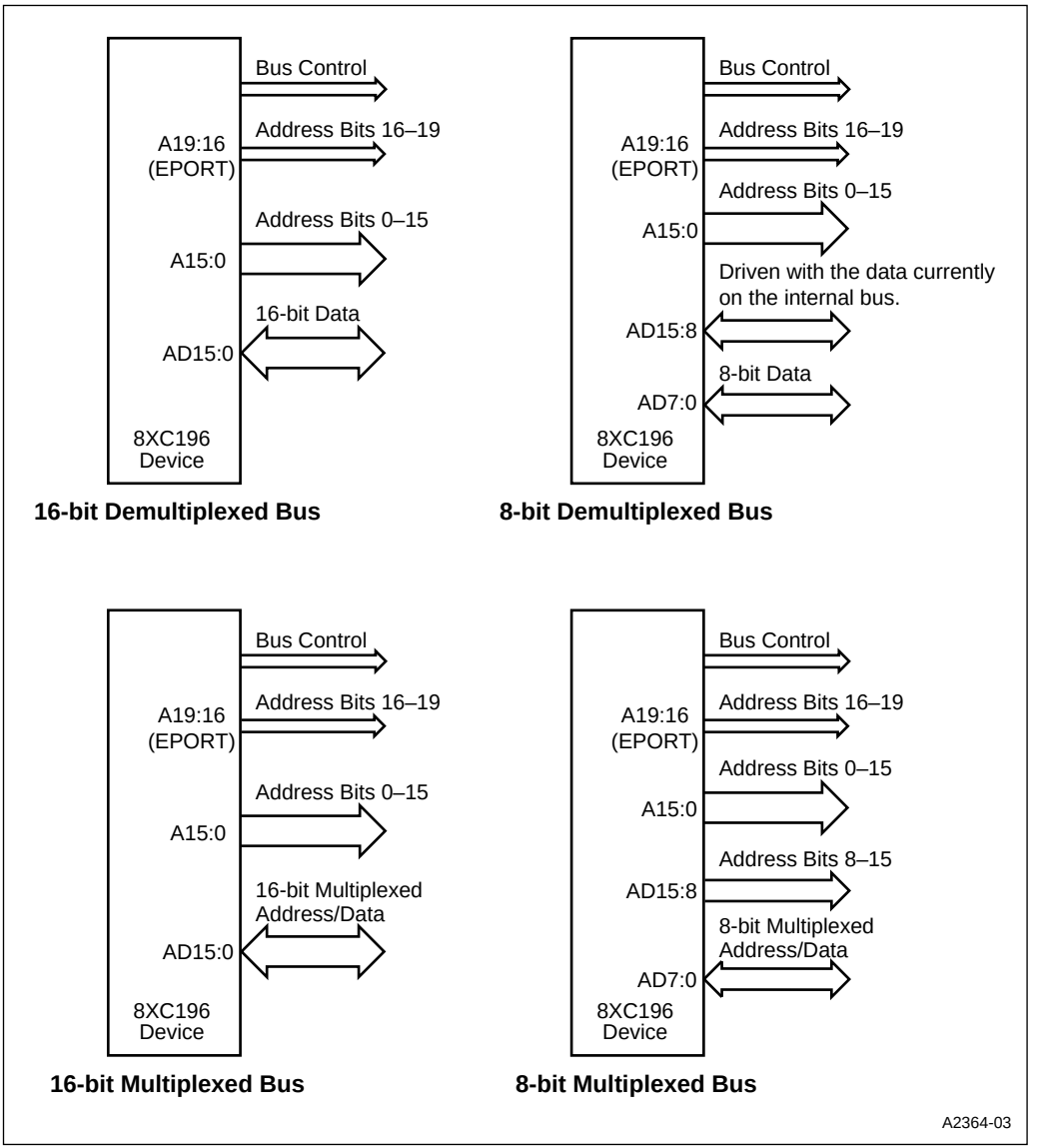


Figure 13-8. Multiplexing and Bus Width Options

A design can incorporate external devices that operate with different bus widths and multiplexing. The bus parameters used during a particular bus cycle are determined by the chip-select output that is assigned to the address being accessed. Figure 13-9 shows the address and data bus configurations for the four combinations of bus width and multiplexing. For detailed waveforms, see "16-bit Bus Timings" on page 13-22 and "System Bus AC Timing Specifications" on page 13-36.

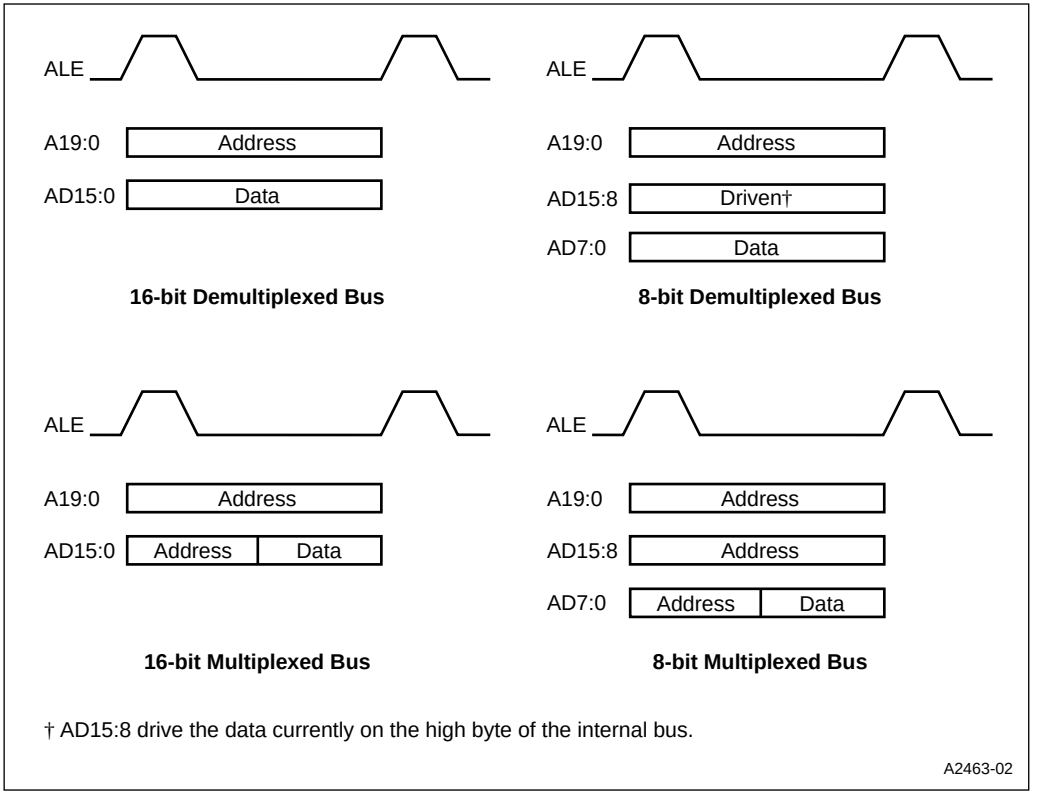


Figure 13-9. Bus Activity for Four Types of Buses

In an 8- or 16-bit demultiplexed mode (top of Figure 13-8 and Figure 13-9), the external device receives the address from A19:0. In a 16-bit system, the data is on AD15:0. In an 8-bit system, the data is on AD7:0. AD15:8 drive the data currently on the high byte of the internal bus.

In multiplexed mode (bottom half of Figure 13-8 and Figure 13-9), both A19:0 and AD15:0 drive the address. A19:0 drive the address throughout the entire bus cycle. For a 16-bit bus width, AD15:0 drive the address for the first half of the bus cycle and drive or receive data during the second half. In the 8-bit case, AD15:8 drive the address during the entire bus cycle.

In multiplexed mode, with the full address on the bus for only half of the cycle, the external device has less time to receive it and to respond. As a result, for the same bus-cycle length (4t) a multiplexed system requires a faster external device (unless wait states are added to the bus cycle). Although the multiplexed mode has this disadvantage, it is useful for compatibility with devices designed for multiplexed operation.

In a 16-bit system (left side of Figure 13-8 and Figure 13-9) one data word can be transferred over AD15:0 in a single bus cycle. In an 8-bit system, one data word is transferred as two bytes over AD7:0 in successive bus cycles, and AD15:8 drive the upper eight address bits for the entire bus cycle.

The flexibility of the chip-select unit enables you to specify the bus width, the number of wait states, and a multiplexed or demultiplexed bus for each of the six chip-select outputs. The system in Figure 13-5 on page 13-13 illustrates a mixture of 8-bit and 16-bit devices with different numbers of wait states.

13.5.1 A 16-bit Example System

Figure 13-10 shows a 16-bit system in demultiplexed mode. The flash memory receives the address on A18:1; data is transferred on AD15:0. Using the WR# signal as shown, this system writes words and not single bytes to the memory. (Using WRL# and WRH#, you can write single bytes on a 16-bit bus.

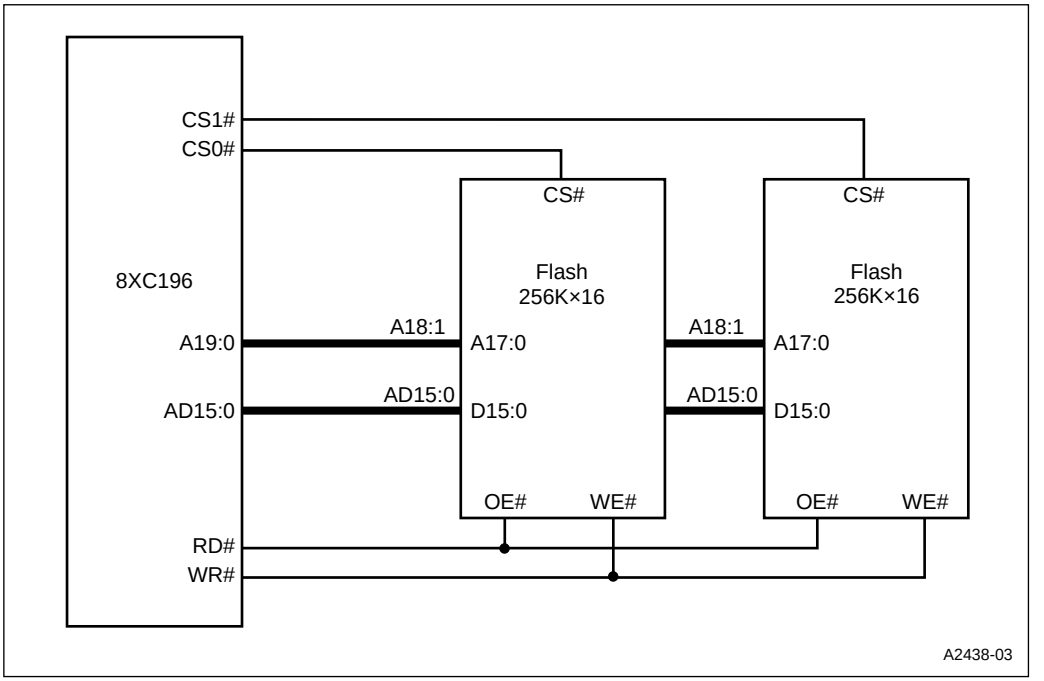


Figure 13-10. 16-bit External Devices in Demultiplexed Mode

13.5.2 16-bit Bus Timings

Figure 13-11 shows idealized 16-bit external-bus timings for the 8XC196NP. The signals are divided into two groups: signals for a demultiplexed bus (top) and signals for a multiplexed bus (bottom). Several bus signals are omitted from the figure to focus on a comparison of multiplexed and demultiplexed buses. The timing parameters are addressed in "Comparison of Multiplexed and Demultiplexed Buses" on page 13-26. Comprehensive timing specifications for both the 8XC196NP and the 80C196NU are shown in Figures 13-20 through 13-23.

CLKOUT and ALE are the same in multiplexed and demultiplexed buses. The CLKOUT period is twice the internal oscillator period (2t). The bus cycles shown here, which have no wait states, require two CLKOUT periods (two state times).

The rising edge of the address latch enable (ALE) indicates that the device is driving an address onto the bus (A19:16 and AD15:0). The device presents a valid address before ALE falls. In a multiplexed system, the ALE signal is used to strobe a transparent latch (such as a 74AC373), which captures the address from AD15:0 and holds it while the bus controller puts data onto AD15:0.

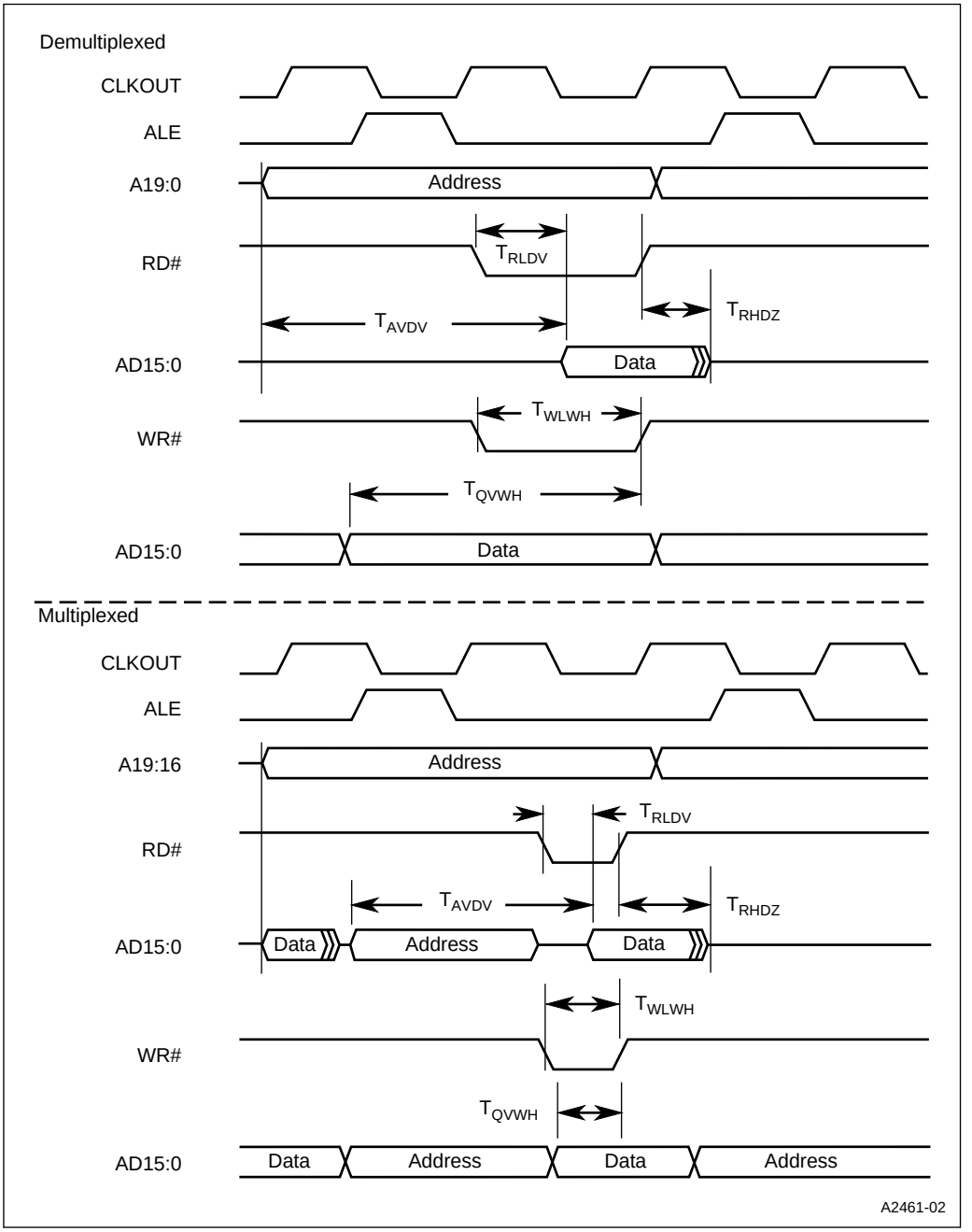


Figure 13-11. Timings for Multiplexed and Demultiplexed 16-bit Buses (8XC196NP)

13.5.3 8-bit Bus Timings

Figure 13-12 shows idealized 8-bit timings for the 8XC196NP. One cycle is required for an 8-bit read or write. A 16-bit access requires two cycles. The first cycle accesses the lower byte, and the second cycle accesses the upper byte. Except for requiring an extra cycle to write the bytes separately, the timings are the same as on the 16-bit bus, and the comparison between the multiplexed and demultiplexed cases is also the same. The demultiplexed bus can accommodate slower memory devices than the multiplexed bus can.

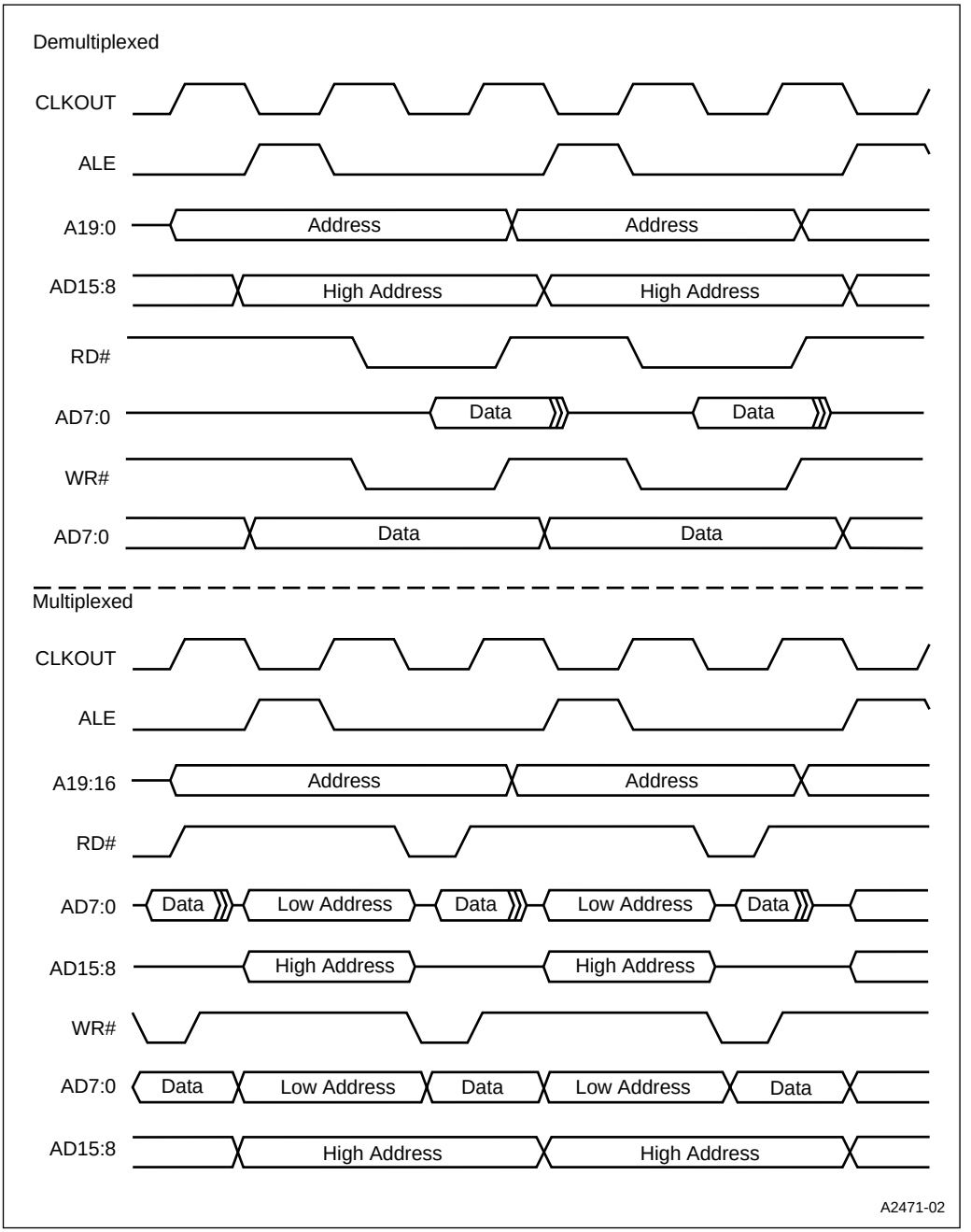


Figure 13-12. Timings for Multiplexed and Demultiplexed 8-bit Buses (8XC196NP)

13.5.4 Comparison of Multiplexed and Demultiplexed Buses

This section compares the timings for multiplexed and demultiplexed buses. A 16-bit bus is used for the comparison. "8-bit Bus Timings" on page 13-24 compares the 8-bit and 16-bit buses.

In a multiplexed system, where AD15:0 carry both address and data, bus activities are time-compressed in comparison with a demultiplexed system, where the address and data have separate lines (A19:0 and AD15:0). The compression is reflected in differences in specifications for the demultiplexed and multiplexed bus. Table 13-10 lists several bus specifications and their values for demultiplexed and multiplexed buses. The data shows that the demultiplexed bus can accommodate slower memory devices. (See "System Bus AC Timing Specifications" on page 13-36 for a complete list of AC timing definitons.)

Table 13-10. Comparison of AC Timings for Demultiplexed and Multiplexed 16-bit Buses

Bus Spec.	Description	Demultiplexed Bus (ns)†	Multiplexed Bus (ns)†
T_{RLDV}	Max. time from RD# asserted to valid input data on the bus.	2t – 25	t – 20
T_{AVDV}	Max. time from A19:0 and CS*x*# valid to valid input data on the bus.	4t – 50	3t – 40
T_{RHDZ}	Max. time from RD# deasserted until data bus is at high impedance.	t	t
T_{WLWH}	Minimum time that WR# is asserted.	2t – 10	t – 5
T_{QVWH}	Minimum time from valid data on the bus to WR# deasserted.	3t – 33	t – 15

† Consult the device datasheet for the latest specifications.

13.6 WAIT STATES (READY CONTROL)

An external device can use the READY input to request wait states in addition to the wait states that are generated internally by the 8XC196N*x* device. When an address is placed on the bus for an external bus cycle, the external device can pull the READY signal low to indicate it is not ready. In response, the bus controller inserts wait states to lengthen the bus cycle until the external device raises the READY signal. Each wait state adds one CLKOUT period (i.e., one state time or 2t) to the bus cycle.

The READY signal is effective for all bus cycles, including the CCB0 fetch (which has three internal wait states). Bits WS0 and WS1 in CCB0 specify the wait states for the CCB1 fetch. Thereafter, the WS0 and WS1 bits in the BUSCON*x* registers control the wait states, and the READY signal can be used to insert additional wait states. (See "Controlling Wait States, Bus Width, and Bus Multiplexing" on page 13-10.)

When selecting infinite wait states, be sure to add external hardware to count wait states and release READY within a specified period of time. Otherwise, a defective external device could tie up the address/data bus indefinitely.

NOTE

Ready control is valid only for external memory; you cannot add wait states when accessing internal ROM.

Setup and hold timings must be met when using the READY signal to insert wait states into a bus cycle (see Table 13-11 and Figures 13-13 through 13-15). Because a decoded, valid address is used to generate the READY signal, the setup time is specified relative to the address being valid. This specification, T_{AVYV}, indicates how much time the external device has to decode the address and assert READY after the address is valid. The READY signal must be held valid until the T_{CLYX} timing specification is met. Typically, this is a minimum of 0 ns from the time CLKOUT goes low. Do not exceed the maximum T_{CLYX} specification or additional (unwanted) wait states might be added. In all cases, refer to the datasheets for the current specifications for T_{AVYV} and T_{CLYX}.

Table 13-11. READY Signal Timing Definitions

Symbol	Definition
T_{AVDV}	Address Valid to Input Data Valid Maximum time the memory device has to output valid data after the device outputs a valid address.
T_{AVYV}	Address Valid to READY Setup Maximum time the memory system has to assert READY after the device outputs the address to guarantee that at least one wait state will occur.
T_{CHYX}	READY Hold after CLKOUT High If maximum specification is exceeded, additional wait states will occur.
T_{CLYX}	READY Hold after CLKOUT Low Minimum hold time is always 0 ns. If maximum specification is exceeded, additional wait states will occur.
T_{LHLH}	ALE Cycle Time Minimum time between ALE pulses.
T_{RLDV}	RD# Low to Input Data Valid Maximum time the memory system has to output valid data after the device asserts RD#.
T_{RLRH}	RD# Low to RD# High RD# pulse width.
T_{QVWH}	Data Valid to WR# High Time between data being valid on the bus and WR# going inactive. Memory devices must meet this specification.
T_{WLWH}	WR# Low to WR# High WR# pulse width.

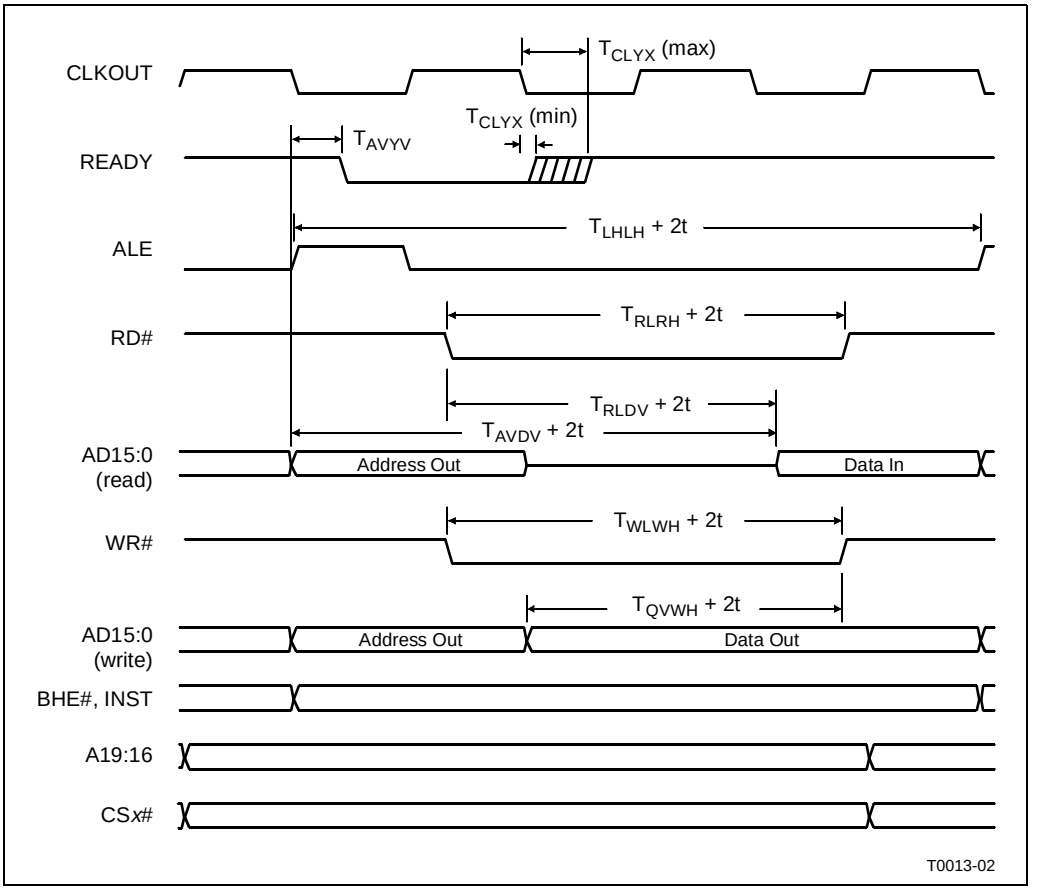


Figure 13-13. READY Timing Diagram — Multiplexed Mode

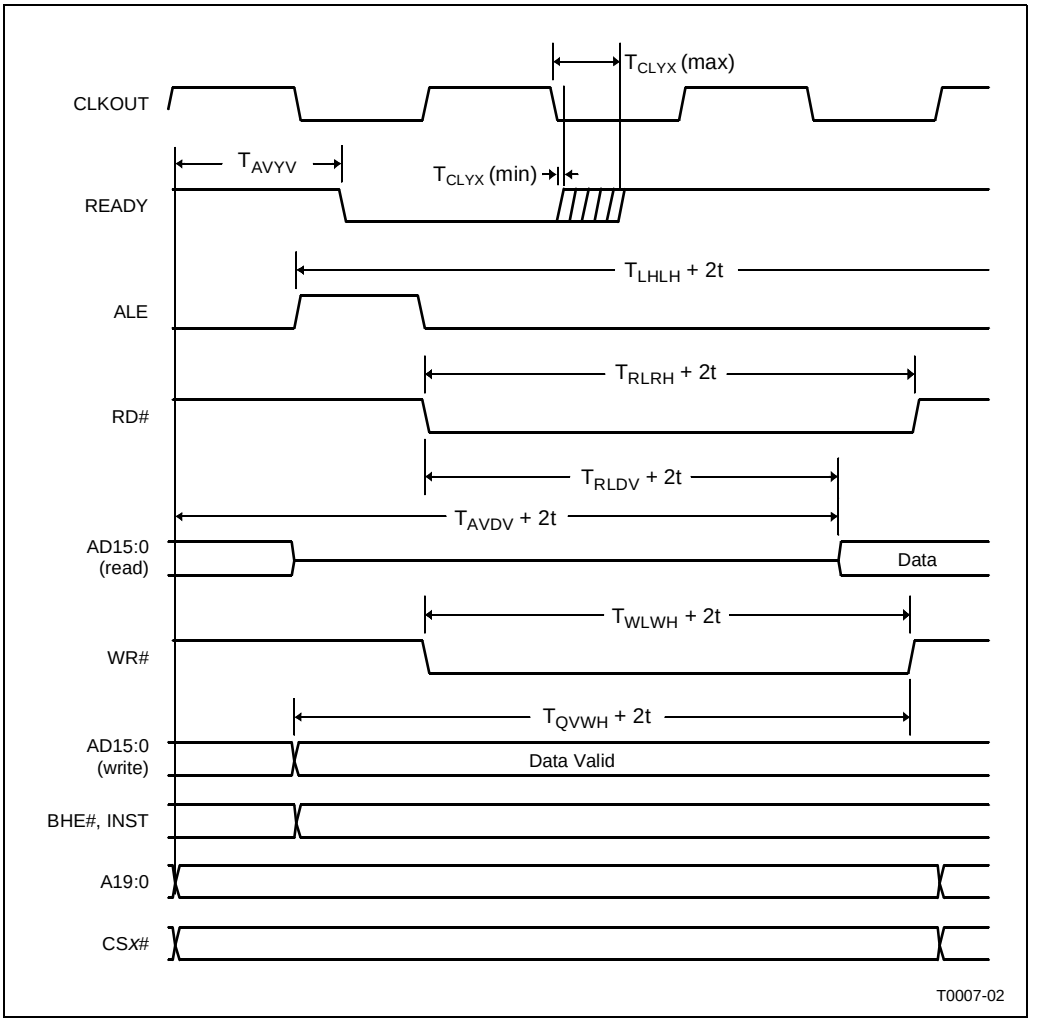


Figure 13-14. READY Timing Diagram — Demultiplexed Mode (8XC196NP)

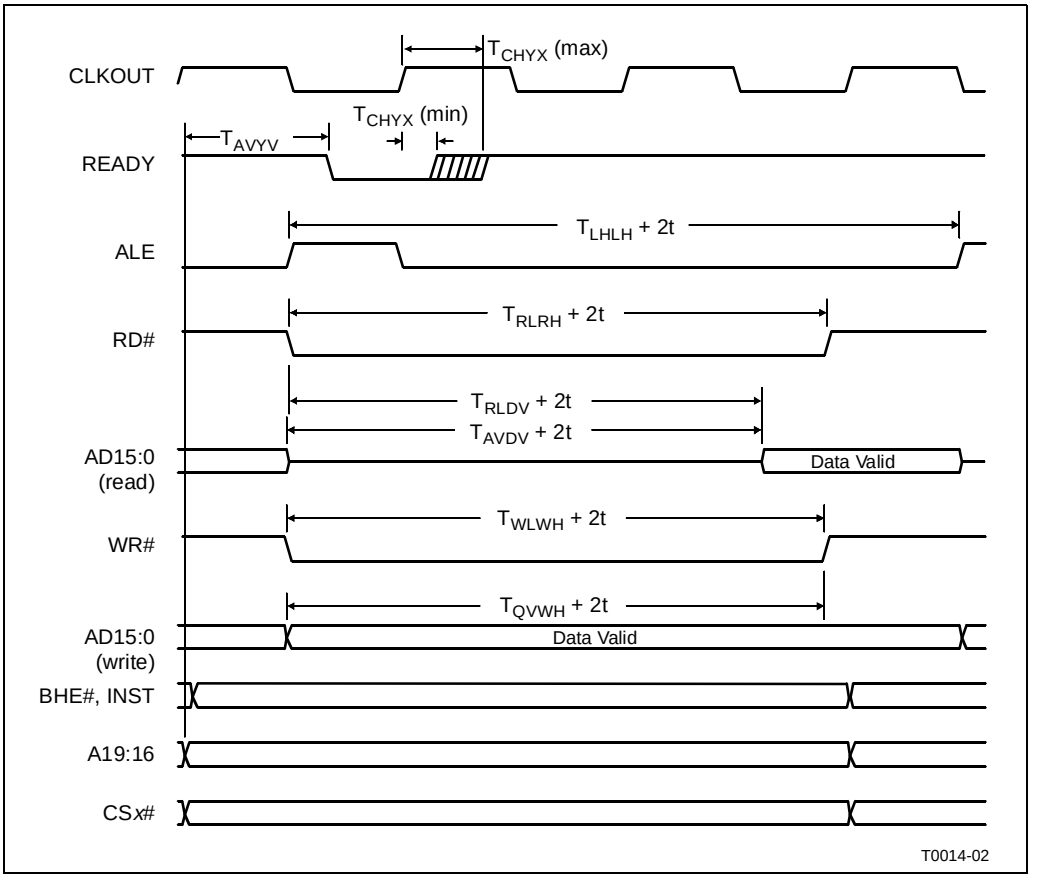


Figure 13-15. READY Timing Diagram — Demultiplexed Mode (80C196NU)

13.7 BUS-HOLD PROTOCOL

The 8XC196N*x* supports a bus-hold protocol that allows external devices to gain control of the address/data bus. The protocol uses three signals, all of which are port 2 special functions: HOLD#/P2.5 (bus-hold request), HLDA#/P2.6 (bus-hold acknowledge), and BREQ#/P2.3 (bus request). When an external device wants to use the 8XC196N*x* bus, it asserts the HOLD# signal. HOLD# is sampled while CLKOUT is low. The 8XC196N*x* responds by releasing the bus and asserting HLDA#. During this hold time, the address/data bus floats, and signals CS*x*#, ALE, RD#, WR#/WRL#, BHE#/WRH#, and INST are weakly held in their inactive states. Figure 13-16 shows the timing for bus-hold protocol, and Table 13-12 on page 13-31 lists the timing parameters and their definitions. Refer to the datasheet for timing parameter values.

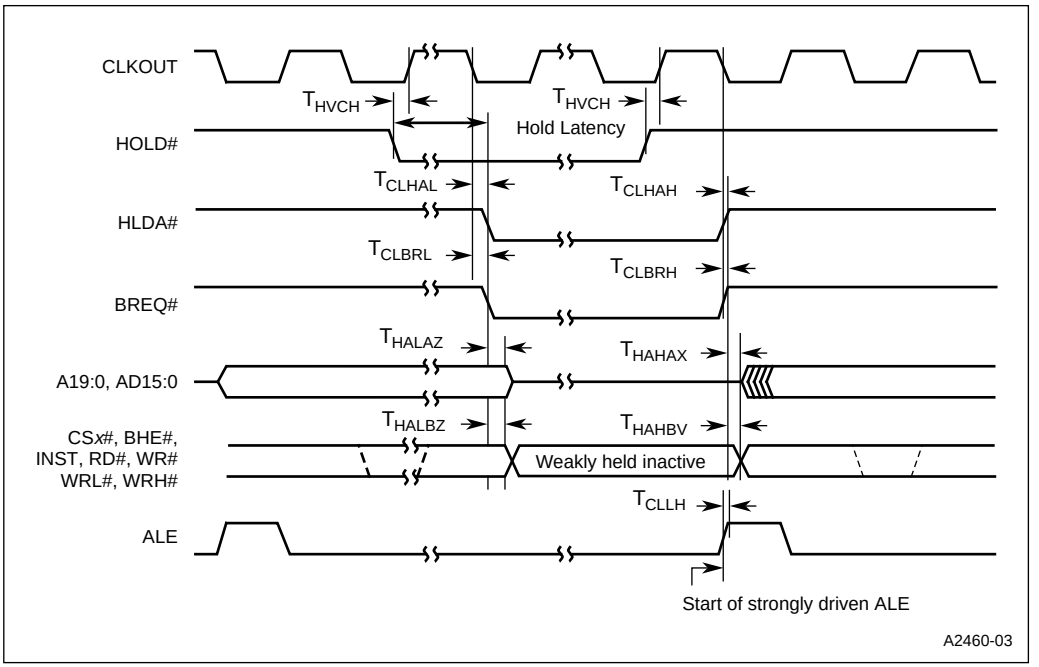


Figure 13-16. HOLD#, HLDA# Timing

Table 13-12. HOLD#, HLDA# Timing Definitions

Symbol	Parameter
T_{HVCH}	HOLD# Setup Time
T_{CLHAL}	CLKOUT Low to HLDA# Low
T_{CLHAH}	CLKOUT Low to HLDA# High
T_{CLBRL}	CLKOUT Low to BREQ# Low
T_{CLBRH}	CLKOUT Low to BREQ# High
T_{HALAZ}	HLDA# Low to Address Float
T_{HAHAX}	HLDA# High to Address No Longer Float
T_{HALBZ}	HLDA# Low to BHE#, INST, RD#, WR#, WRL#, WRH# Weakly Driven
T_{HAHBV}	HLDA# High to BHE#, INST, RD#, WR#, WRL#, WRH# valid
T_{CLLH}	Clock Falling to ALE Rising; Use to derive other timings.

When the external device is finished with the bus, it relinquishes control by driving HOLD# high. In response, the 8XC196N*x* deasserts HLDA# and resumes control of the bus.

If the 8XC196N*x* has a pending external bus cycle while it is in hold (another device has control of the bus), it asserts BREQ# to request control of the bus. After the external device responds by releasing HOLD#, the 8XC196N*x* exits hold and then deasserts BREQ# and HLDA#.

13.7.1 Enabling the Bus-hold Protocol

To use the bus-hold protocol, you must configure P2.3/BREQ#, P2.5/HOLD#, and P2.6/HLDA# to operate as special-function signals. BREQ# and HLDA# are active-low outputs; HOLD# is an active-low input.

You must also set the hold enable bit (HLDEN) in the window selection register (WSR.7) to enable the bus-hold protocol. Once the bus-hold protocol has been selected, the port functions of P2.3, P2.5, and P2.6 cannot be selected without resetting the device. (During the time that the pins are configured to operate as special-function signals, their special-function values can be read from the P2_PIN.*x* bits.) However, the hold function can be dynamically enabled and disabled as described in "Disabling the Bus-hold Protocol."

13.7.2 Disabling the Bus-hold Protocol

To disable hold requests, clear WSR.7. The 8XC196N*x* does not take control of the bus immediately after HLDEN is cleared. Instead, it waits for the current hold request to finish and then disables the bus-hold feature and ignores any new requests until the bit is set again.

Sometimes it is important to prevent another device from taking control of the bus while a block of code is executing. One way to protect a code segment is to clear WSR.7 and then execute a JBC instruction to check the status of the HLDA# signal. The JBC instruction prevents the RALU from executing the protected block until current hold requests are serviced and the hold feature is disabled. This is illustrated in the following code:

```
          DI                          ;Disable interrupts to prevent
                                      ;code interruption
          PUSH WSR                    ;Disable hold requests and
          LDB  WSR,#1FH               ;window Port 2
WAIT:     JBC  P2_PIN,6, WAIT         ;Check the HLDA# signal. If set,
                                      ;add protected instruction here
          POP  WSR                    ;Enable hold requests
          EI                          ;Enable interrupts
```

13.7.3 Hold Latency

When an external device asserts HOLD#, the 8XC196N*x* finishes the current bus cycle and then asserts HLDA#. The time it takes the device to assert HLDA# after the external device asserts HOLD# is called *hold latency* (see Figure 13-16 on page 13-31). Table 13-13 lists the maximum hold latency for each type of bus cycle.

Table 13-13. Maximum Hold Latency

Bus Cycle Type	Maximum Hold Latency (state times)
Internal execution or idle mode	1.5
16-bit external execution	2.5 + 1 per wait state
8-bit external execution	2.5 + 2 per wait state

13.7.4 Regaining Bus Control

While HOLD# is asserted, the 8XC196N*x* continues executing code until it needs to access the external bus. If executing from internal memory, it continues until it needs to perform an external memory cycle. If executing from external memory, it continues executing until the queue is empty or until it needs to perform an external data cycle. As soon as it needs to access the external bus, the 8XC196N*x* asserts BREQ# and waits for the external device to deassert HOLD#. After asserting BREQ#, the 8XC196N*x* cannot respond to any interrupt requests, including NMI, until the external device deasserts HOLD#. One state time after HOLD# goes high, the 8XC196N*x* deasserts HLDA# and, with no delay, resumes control of the bus.

If the 8XC196N*x* is reset while in hold, bus contention can occur. For example, a CPU-only device would try to fetch the chip configuration byte from external memory after RESET# was brought high. Bus contention would occur because both the external device and the 8XC196N*x* would attempt to access memory. One solution is to use the RESET# signal as the system reset; then all bus masters (including the 8XC196N*x*) are reset at once. Chapter 11, “Minimum Hardware Considerations,” shows system reset circuit examples.

13.8 WRITE-CONTROL MODES

The device has two write-control modes: the standard mode, which uses the WR# and BHE# signals, and the write strobe mode, which uses the WRL# and WRH# signals. Otherwise, the two modes are identical. The modes are selected by chip configuration register 0 (Figure 13-6 on page 13-15.)

Figure 13-17 shows the waveforms of the asserted write-control signals in the two modes. Note that only BHE# is valid throughout the bus cycle.

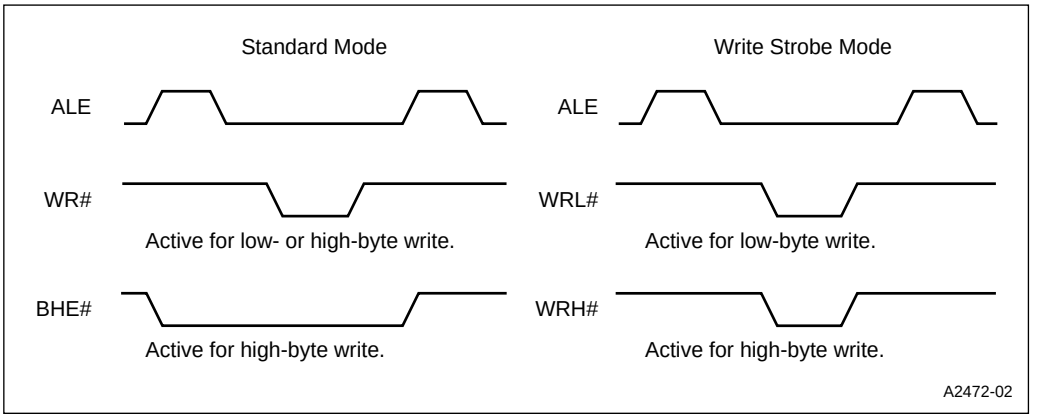


Figure 13-17. Write-control Signal Waveforms

Table 13-14 compares the values of the write-control signals for write operations in the standard mode and the write strobe mode. The table lists values of WR# and BHE# and values of WRL# and WRH# for 8-bit and 16-bit writes on an 8-bit and 16-bit bus.

Table 13-14. Write Signals for Standard and Write Strobe Modes

Bus Width	Word/Byte Written	A0	Standard (CCR0.2 = 1)		Write Strobe (CCR0.2 = 0)	
			WR#	BHE#	WRL#	WRH#
8	Low Byte	0	0	1	0	0
	High Byte	1	0	0	0	0
	Word	0	0	0	0	0
		1	Illegal		Illegal	
16	Low Byte	0	0	1	0	1
	High Byte	1	0	0	1	0
	Word	0	0	0	0	0
		1	Illegal		Illegal	

To select the standard write-control mode, set CCR0.2. In standard mode, the WR#/WRL# pin operates as WR#, and the BHE#/WRH# pin operates as BHE#. WR# is asserted for every external memory write. BHE# is asserted for word accesses (read and write) and for byte accesses to odd addresses. BHE# can be used to select the bank of memory that stores the high (odd) byte. Figure 13-10 on page 13-22 illustrates use of the standard mode in a 16-bit system. In this example, WR# writes words to the 16-bit flash memory. To write individual bytes, you can use the decoding logic in Figure 13-18 or use the write strobe mode.

To write single bytes on a 16-bit bus requires separate low-byte and high-byte write signals (WRL# and WRH#). Figure 13-18 shows a sample circuit that combines WR#, BHE#, and address bit 0 (A0) to produce these signals. This additional logic is unnecessary, however. In the write strobe mode, WRL# and WRH# are available at the device's external pins.

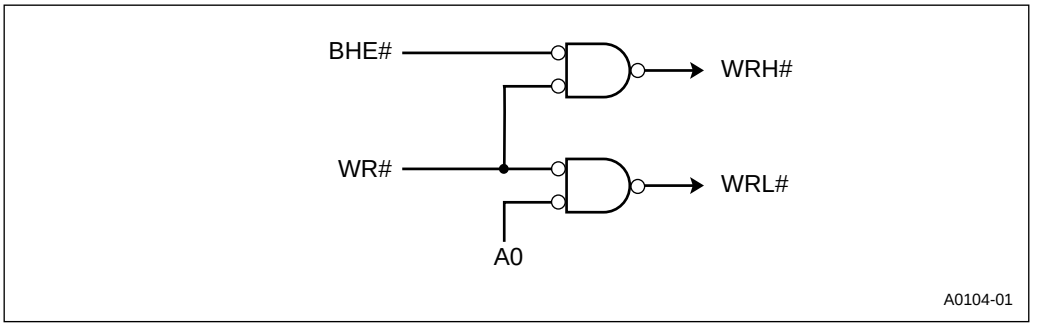


Figure 13-18. Decoding WRL# and WRH#

The write strobe mode eliminates the need to externally decode high-byte and low-byte write signals to external 16-bit memory on a 16-bit bus. When the write strobe mode is selected, the WR#/WRL# pin operates as WRL#, and the BHE#/WRH# pin operates as WRH#. In the 16-bit bus mode, WRL# is asserted for all low-byte writes (even addresses) and all word writes, and WRH# is asserted for all high-byte writes (odd addresses) and all word writes. In the 8-bit bus mode, WRH# and WRL# are asserted for both even and odd addresses (see Table 13-14).

Figure 13-19 illustrates the use of the write strobe mode in a mixed 8-bit and 16-bit system with two flash memories and one SRAM. The WRL# signal, which is generated for all 8-bit writes (Table 13-14), is used to write bytes to the SRAM. Note that the RD# signal is sufficient for single-byte reads on a 16-bit bus. Both bytes are put onto the data bus and the memory controller discards the unwanted byte.

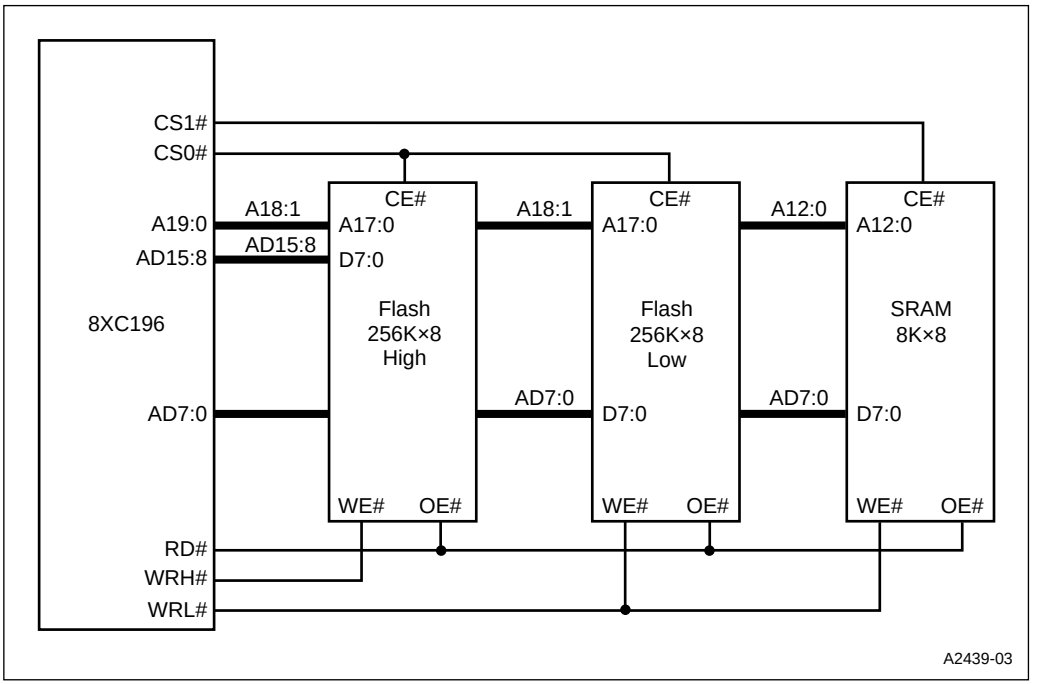


Figure 13-19. A System with 8-bit and 16-bit Buses

13.9 SYSTEM BUS AC TIMING SPECIFICATIONS

Refer to the latest datasheet for the AC timings to make sure your system meets specifications. The major external bus timing specifications are shown in Figure 13-20 through 13-23.

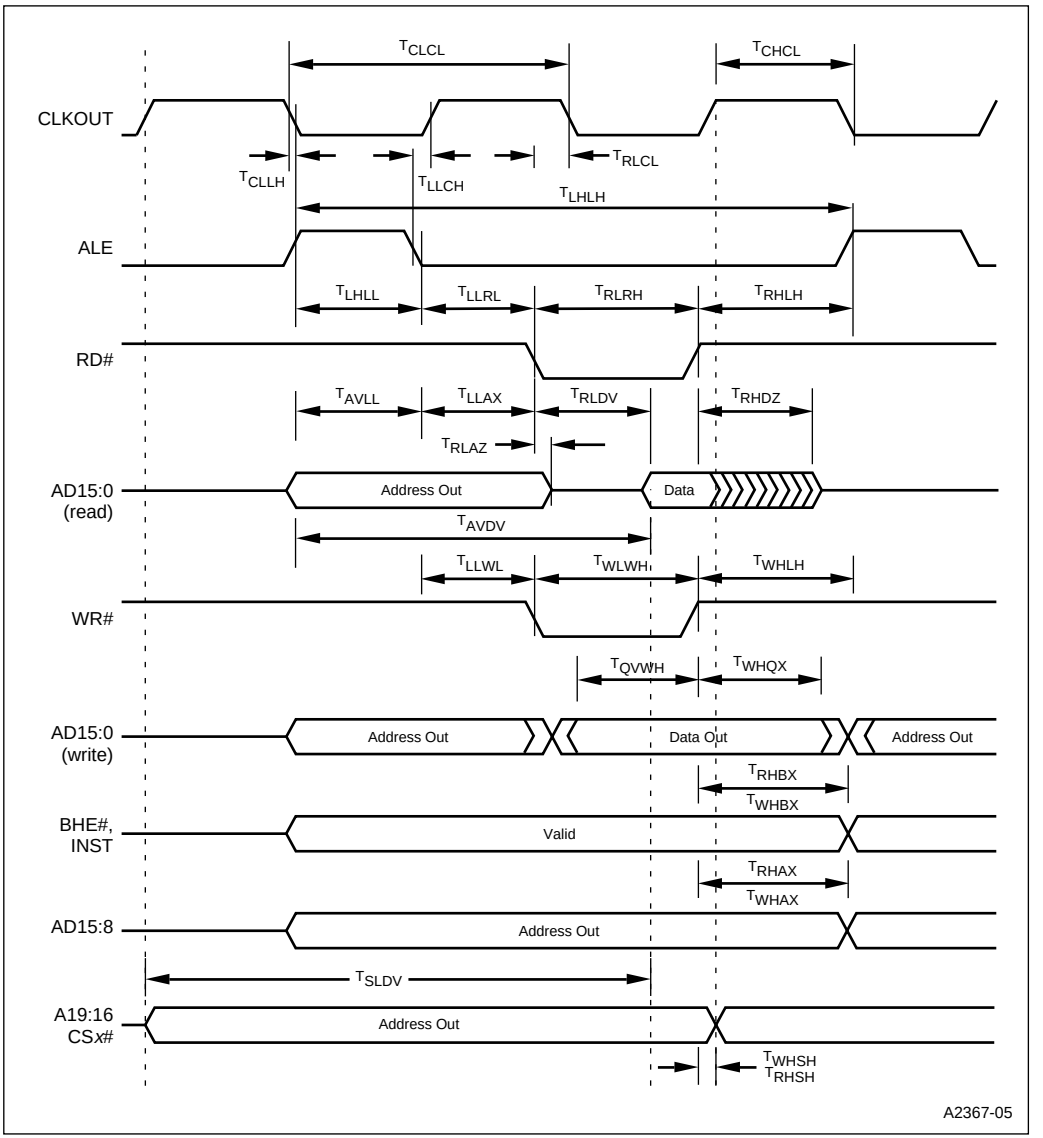


Figure 13-20. Multiplexed System Bus Timing (8XC196NP)

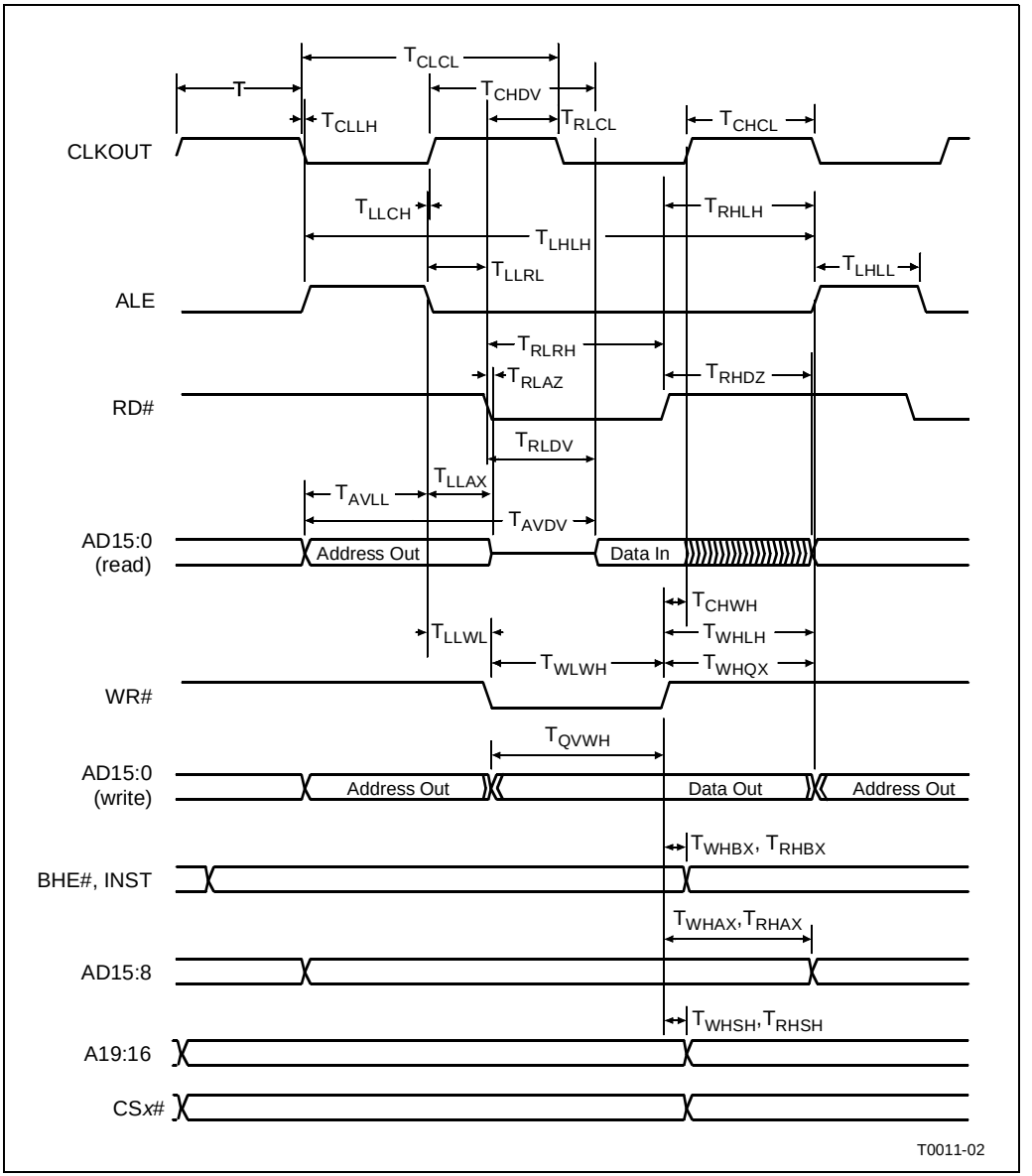


Figure 13-21. Multiplexed System Bus Timing (80C196NU)

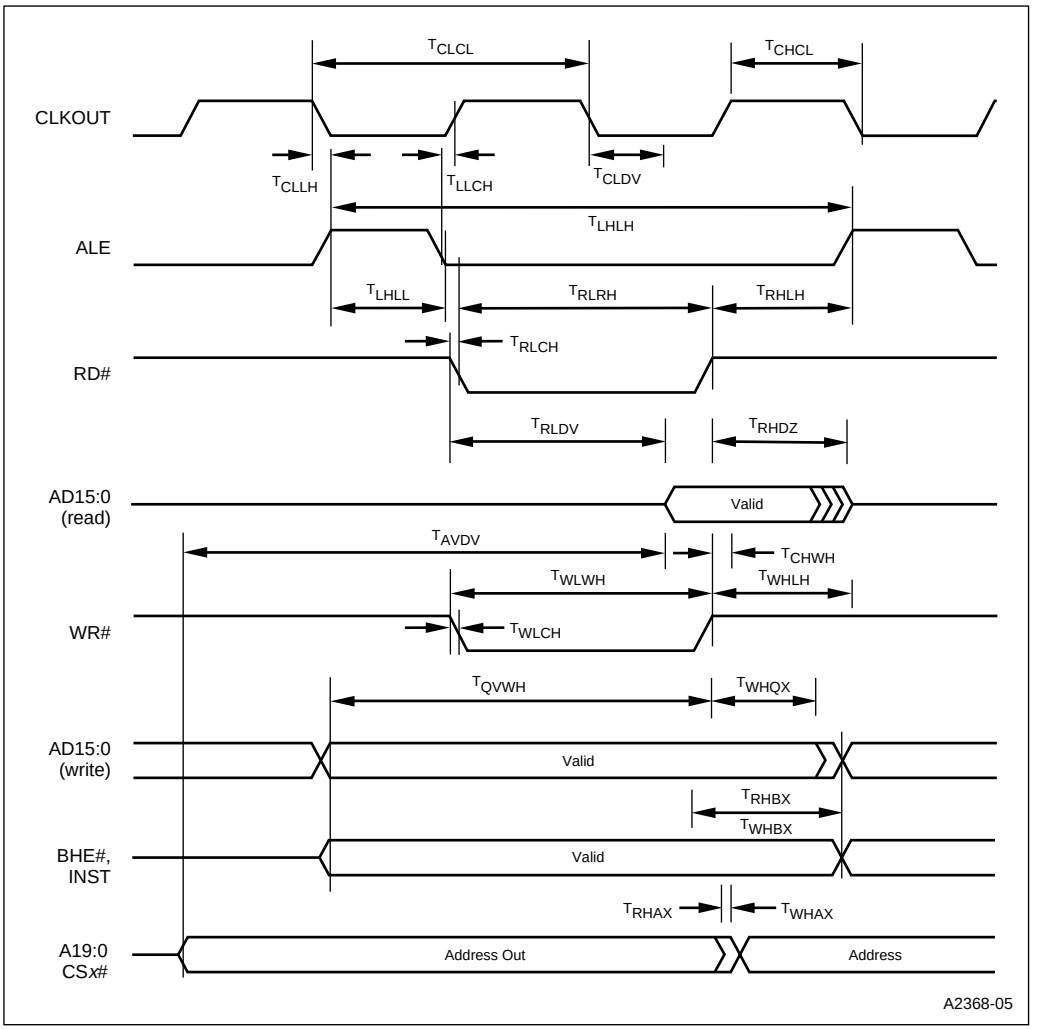


Figure 13-22. Demultiplexed System Bus Timing (8XC196NP)

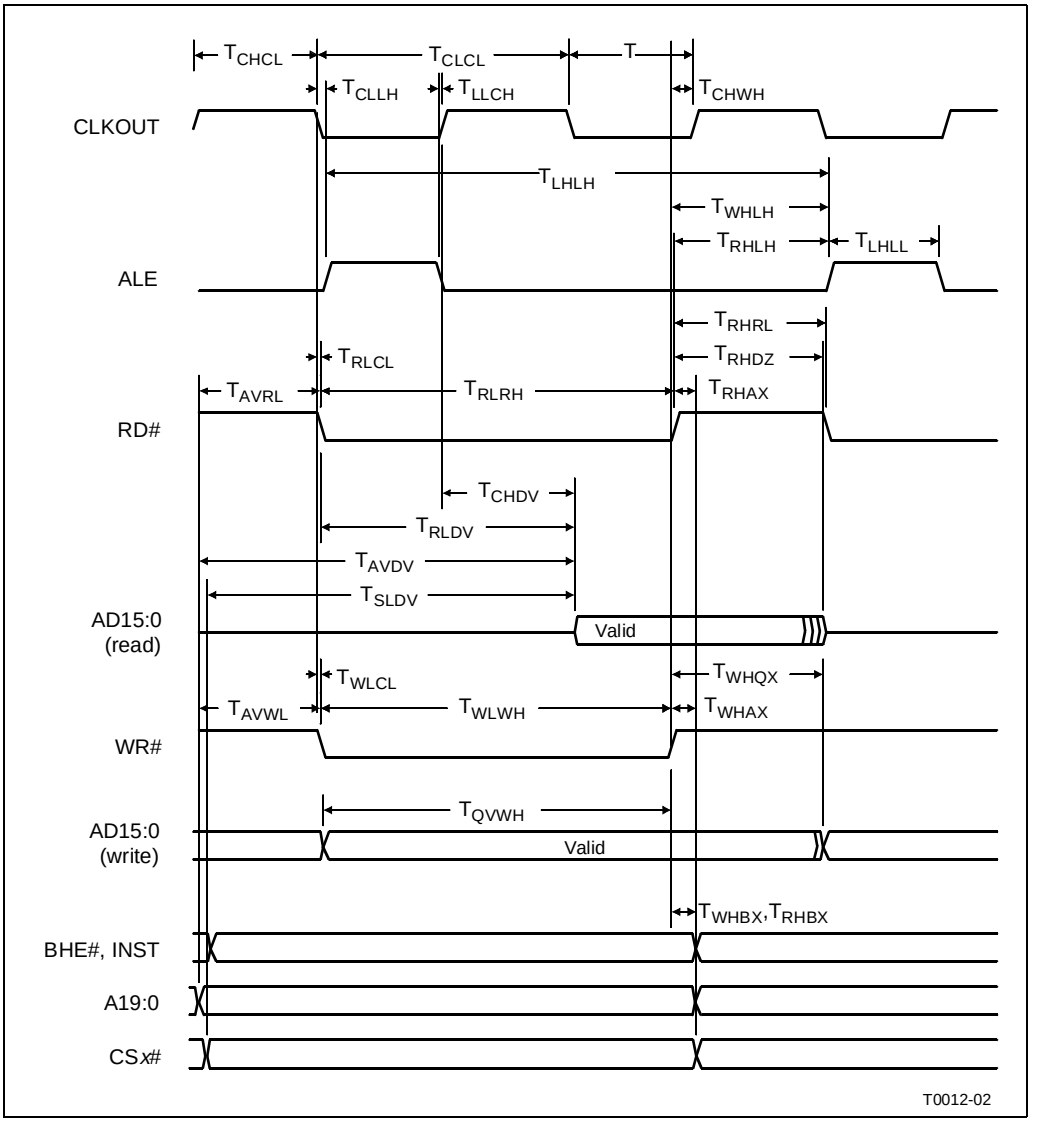


Figure 13-23. Demultiplexed System Bus Timing (80C196NU)

13.9.1 Deferred Bus-cycle Mode (80C196NU Only)

The 80C196NU offers a deferred bus cycle mode. This bus mode (enabled by CCR1.5; see Figure 13-7 on page 13-16) reduces bus contention when using the 80C196NU in demultiplexed mode with slow memories. As shown in Figure 13-24, a delay of 2t occurs in the first bus cycle following a chip-select output change and the first write cycle following a read cycle.

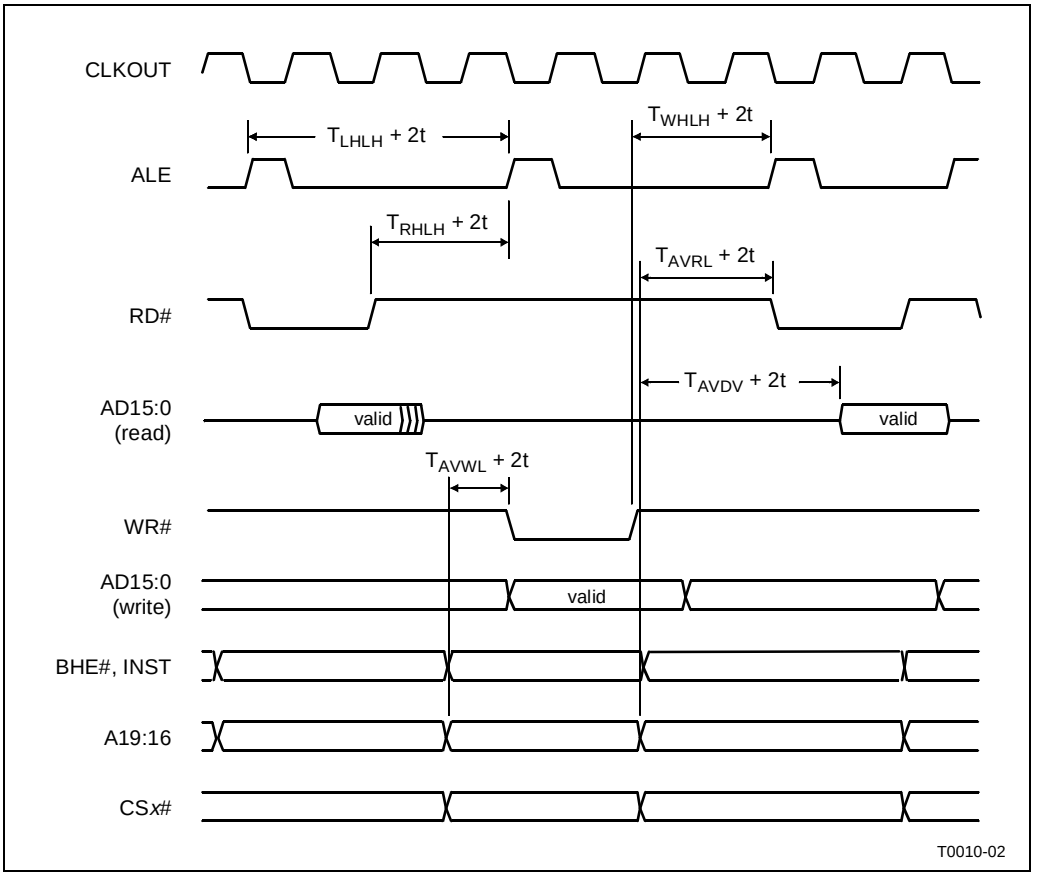


Figure 13-24. Deferred Bus-cycle Mode Timing Diagram (80C196NU)

13.9.2 Explanation of AC Symbols

Each symbol consists of two pairs of letters prefixed by "T" (for time). The characters in a pair indicate a signal and its condition, respectively. Symbols represent the time between the two signal/condition points. For example, T_{LLRL} is the time between signal L (ALE) condition L (Low) and signal R (RD#) condition L (Low). Table 13-15 defines the signal and condition codes.

Table 13-15. AC Timing Symbol Definitions

Signals						Conditions	
A†	Address	H	HOLD#	S	CS*x*#	H	High
B	BHE#	HA	HLDA#	W	WR#, WRH#, WRL#	L	Low
C	CLKOUT	L	ALE	X	XTAL1	V	Valid
D	Data	Q	Data Out	Y	READY	X	No Longer Valid
G	Buswidth	R	RD#			Z	Floating

† Address bus (demultiplexed mode) or address/data bus (multiplexed mode)

13.9.3 AC Timing Definitions

Table 13-16 defines the AC timing specifications that the memory system must meet and those that the device will provide.

Table 13-16. AC Timing Definitions

Symbol	Definition
The External Memory System Must Meet These Specifications	
T_{AVDV}	Address Valid to Input Data Valid Maximum time the memory device has to output valid data after the device outputs a valid address.
T_{CHDV}	CLKOUT High to Input Data Valid Maximum time the memory system has to output valid data after CLKOUT rises.
T_{CLDV}	CLKOUT Low to Input Data Valid Maximum time the memory system has to output valid data after CLKOUT falls.
T_{QVWH}	Data Valid to WR# High Time between data being valid on the bus and WR# going inactive.
T_{RHDZ}	RD# High to Input Data Float Time after RD# is inactive until the memory system must float the bus. If this timing is not met, bus contention will occur.
T_{RLDV}	RD# Low to Input Data Valid Maximum time the memory system has to output valid data after the device asserts RD#.
T_{SLDV}	CS*x*# Valid to Input Data Valid Maximum time the memory device has to output valid data after the device outputs a valid chip-select output.

Table 13-16. AC Timing Definitions (Continued)

Symbol	Definition
The 8XC196N*x* Meets These Specifications	
f	Operating frequency Frequency of the signal input on the XTAL1 pin times the clock multiplier (*x*). For the 8XC196NP, *x* is always 1; for the 80C196NU, *x* is 1, 2, or 4, depending on the clock mode. The internal bus speed of the device is ½ f.
t	Operating period (1/f) All AC Timings are referenced to t.
T_{AVLL}	Address Setup to ALE Low Length of time ADDRESS is valid before ALE falls. Use this specification when designing the external latch.
T_{AVRL}	Address Setup to RD# Low Length of time ADDRESS is valid before RD# falls.
T_{AVWL}	Address Setup to WR# Low Length of time ADDRESS is valid before WR# falls.
T_{CHCL}	CLKOUT High Period Needed in systems that use CLKOUT as clock for external devices.
T_{CHWL}	CLKOUT High to WR# Low Time between CLKOUT going high and WR# going active.
T_{CLCL}	CLKOUT Cycle Time Normally 2t.
T_{CLLH}	CLKOUT Falling to ALE Rising Use to derive other timings.
T_{LHLH}	ALE Cycle Time Minimum time between ALE pulses.
T_{LHLL}	ALE High Period Use this specification when designing the external latch.
T_{LLAX}	Address Hold after ALE Low Length of time ADDRESS is valid after ALE falls. Use this specification when designing the external latch.
T_{LLCH}	ALE Falling to CLKOUT Rising Use to derive other timings.
T_{LLRL}	ALE Low to RD# Low Length of time after ALE falls before RD# is asserted. Could be needed to ensure proper memory decoding takes place before a device is enabled.
T_{LLWL}	ALE Low to WR# Low Length of time after ALE falls before WR# is asserted. Could be needed to ensure proper memory decoding takes place before a device is enabled.

Table 13-16. AC Timing Definitions (Continued)

Symbol	Definition
The 8XC196N*x* Meets These Specifications (Continued)	
T_{RHAX}	(Multiplexed Mode) AD15:8/CS*x*# Hold after RD# High Minimum time the high byte of the address in 8-bit mode will be valid after RD# inactive. (Demultiplexed Mode) A19:0/CS*x*# Hold after RD# High Minimum time the address will be valid after RD# inactive.
T_{RHBX}	BHE#, INST Hold after RD# High Minimum time these signals will be valid after RD# inactive.
T_{RHLH}	RD# High to ALE Rising Time between RD# going inactive and the next ALE. Useful in calculating time between RD# inactive and next address valid.
T_{RHRL}	RD# High to RD# Low Minimum RD# inactive time.
T_{RHSH}	A19:0/CS*x*# Hold after RD# High Minimum time the address and chip-select output are held after RD# inactive.
T_{RLAZ}	RD# Low to Address Float Used to calculate when the device stops driving address on the bus.
T_{RLCH}	RD# Low to CLKOUT High Maximum time between RD# being asserted and CLKOUT going high.
T_{RLCL}	RD# Low to CLKOUT Low Length of time from RD# asserted to CLKOUT falling edge.
T_{RLRH}	RD# Low to RD# High RD# pulse width.
T_{WHAX}	(Multiplexed Mode) AD15:8/CS*x*# Hold after WR# High Minimum time the high byte of the address in 8-bit mode will be valid after WR# inactive. (Demultiplexed Mode) A19:0/CS*x*# Hold after WR# High Minimum time the address will be valid after WR# inactive.
T_{WHBX}	BHE#, INST Hold after WR# High Minimum time these signals will be valid after WR# inactive.
T_{WHLH}	WR# High to ALE High Time between WR# going inactive and next ALE. Also used to calculate WR# inactive and next Address valid.
T_{WHQX}	Data Hold after WR# High Length of time after WR# rises that the data stays valid on the bus.

Table 13-16. AC Timing Definitions (Continued)

Symbol	Definition
The 8XC196N*x* Meets These Specifications (Continued)	
T_{WHSH}	A19:0/CS*x*# Hold after WR# High Minimum time the address and chip-select output are held after WR# inactive.
T_{WLCH}	WR# Low to CLKOUT High Minimum and maximum time between WR# being asserted and CLKOUT going high.
T_{WLCL}	WR# Low to CLKOUT Low Minimum and maximum time between WR# being asserted and CLKOUT going low.
T_{WLWH}	WR# Low to WR# High WR# pulse width.

A

Instruction Set Reference

APPENDIX A
INSTRUCTION SET REFERENCE

This appendix provides reference information for the instruction set of the family of MCS® 96 microcontrollers. It defines the processor status word (PSW) flags, describes each instruction, shows the relationships between instructions and PSW flags, and shows hexadecimal opcodes, instruction lengths, and execution times. It includes the following tables.

- Table A-1 on page A-2 is a map of the opcodes.
- Table A-2 on page A-4 defines the processor status word (PSW) flags.
- Table A-3 on page A-5 shows the effect of the PSW flags or a specified register bit on conditional jump instructions.
- Table A-4 on page A-5 defines the symbols used in Table A-6.
- Table A-5 on page A-6 defines the variables used in Table A-6 to represent instruction operands.
- Table A-6 beginning on page A-7 lists the instructions alphabetically, describes each of them, and shows the effect of each instruction on the PSW flags.
- Table A-7 beginning on page A-47 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics.
- Table A-8 on page A-53 lists instruction lengths and opcodes for each applicable addressing mode.
- Table A-9 on page A-60 lists instruction execution times, expressed in state times.

NOTE

The # symbol prefixes an immediate value in immediate addressing mode. Chapter 4, “Programming Considerations,” describes the operand types and addressing modes.

Table A-1. Opcode Map (Left Half)

Opcode	*x*0	*x*1	*x*2	*x*3	*x*4	*x*5	*x*6	*x*7
0*x*	SKIP	CLR	NOT	NEG	XCH di	DEC	EXT	INC
1*x*		CLRB	NOTB	NEGB	XCHB di	DECB	EXTB	INCB
2*x*	SJMP							
3*x*	JBC bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
4*x*	AND 3op di	im	in	ix	ADD 3op di	im	in	ix
5*x*	ANDB 3op di	im	in	ix	ADDB 3op di	im	in	ix
6*x*	AND 2op di	im	in	ix	ADD 2op di	im	in	ix
7*x*	ANDB 2op di	im	in	ix	ADDB 2op di	im	in	ix
8*x*	OR di	im	in	ix	XOR di	im	in	ix
9*x*	ORB di	im	in	ix	XORB di	im	in	ix
A*x*	LD di	im	in	ix	ADDC di	im	in	ix
B*x*	LDB di	im	in	ix	ADDCB di	im	in	ix
C*x*	ST di	BMOV	ST in	ix	STB di	CMPL	STB in	ix
D*x*	JNST	JNH	JGT	JNC	JNVT	JNV	JGE	JNE
E*x*	DJNZ	DJNZW	TIJMP	BR/EBR in	EBMOVI		EJMP	LJMP
F*x*	RET	ECALL	PUSHF	POPF	PUSHA	POPA	IDLPD	TRAP

NOTE: The first digit of the opcode is listed vertically, and the second digit is listed horizontally. The related instruction mnemonic is shown at the intersection of the two digits. Shading indicates reserved opcodes. If the CPU attempts to execute an unimplemented opcode, an interrupt occurs. For more information, see "Unimplemented Opcode" on page 6-5.

Table A-1. Opcode Map (Right Half)

Opcode	*x*8	*x*9	*x*A	*x*B	*x*C	*x*D	*x*E	*x*F
0*x*	SHR	SHL	SHRA	XCH ix	SHRL	SHLL	SHRAL	NORML
1*x*	SHRB	SHLB	SHRAB	XCHB ix	EST in	EST ix	ESTB in	ESTB ix
2*x*	SCALL							
3*x*	JBS bit 0	JBS bit 1	JBS bit 2	JBS bit 3	JBS bit 4	JBS bit 5	JBS bit 6	JBS bit 7
4*x*	SUB 3op di	SUB 3op im	SUB 3op in	SUB 3op ix	MULU 3op (Note 2) di	MULU 3op (Note 2) im	MULU 3op (Note 2) in	MULU 3op (Note 2) ix
5*x*	SUBB 3op di	SUBB 3op im	SUBB 3op in	SUBB 3op ix	MULUB 3op (Note 2) di	MULUB 3op (Note 2) im	MULUB 3op (Note 2) in	MULUB 3op (Note 2) ix
6*x*	SUB 2op di	SUB 2op im	SUB 2op in	SUB 2op ix	MULU 2op (Note 2) di	MULU 2op (Note 2) im	MULU 2op (Note 2) in	MULU 2op (Note 2) ix
7*x*	SUBB 2op di	SUBB 2op im	SUBB 2op in	SUBB 2op ix	MULUB 2op (Note 2) di	MULUB 2op (Note 2) im	MULUB 2op (Note 2) in	MULUB 2op (Note 2) ix
8*x*	CMP di	CMP im	CMP in	CMP ix	DIVU (Note 2) di	DIVU (Note 2) im	DIVU (Note 2) in	DIVU (Note 2) ix
9*x*	CMPB di	CMPB im	CMPB in	CMPB ix	DIVUB (Note 2) di	DIVUB (Note 2) im	DIVUB (Note 2) in	DIVUB (Note 2) ix
A*x*	SUBC di	SUBC im	SUBC in	SUBC ix	LDBZE di	LDBZE im	LDBZE in	LDBZE ix
B*x*	SUBCB di	SUBCB im	SUBCB in	SUBCB ix	LDBSE di	LDBSE im	LDBSE in	LDBSE ix
C*x*	PUSH di	PUSH im	PUSH in	PUSH ix	POP di	BMOVI	POP in	POP ix
D*x*	JST	JH	JLE	JC	JVT	JV	JLT	JE
E*x*	ELD in	ELD ix	ELDB in	ELDB ix	DPTS	EPTS	(Note 1)	LCALL
F*x*	CLRC	SETC	DI	EI	CLRVT	NOP	signed MUL/DIV (Note 2)	RST

NOTES:
1. This opcode is reserved, but it does not generate an unimplemented opcode interrupt.
2. Signed multiplication and division are two-byte instructions. The first byte is “FE” and the second is the opcode of the corresponding unsigned instruction.

Table A-2. Processor Status Word (PSW) Flags

Mnemonic	Description
C	The carry flag is set to indicate an arithmetic carry from the MSB of the ALU or the state of the last bit shifted out of an operand. If a subtraction operation generates a borrow, the carry flag is cleared. **C** **Value of Bits Shifted Off** 0 < ½ LSB 1 ≥ ½ LSB Normally, the result is rounded up if the carry flag is set. The sticky bit flag allows a finer resolution in the rounding decision. **C ST** **Value of Bits Shifted Off** 0 0 = 0 0 1 > 0 and < ½ LSB 1 0 = ½ LSB 1 1 > ½ LSB and < 1 LSB
N	The negative flag is set to indicate that the result of an operation is negative. The flag is correct even if an overflow occurs. For all shift operations and the NORML instruction, the flag is set to equal the most-significant bit of the result, even if the shift count is zero.
ST	The sticky bit flag is set to indicate that, during a right shift, a "1" has been shifted into the carry flag and then shifted out. This bit is undefined after a multiply operation. The sticky bit flag can be used with the carry flag to allow finer resolution in rounding decisions. See the description of the carry (C) flag for details.
V	The overflow flag is set to indicate that the result of an operation is too large to be represented correctly in the available space. For shift operations, the flag is set if the most-significant bit of the operand changes during the shift. For divide operations, the quotient is stored in the low-order half of the destination operand and the remainder is stored in the high-order half. The overflow flag is set if the quotient is outside the range for the low-order half of the destination operand. (Chapter 4, "Programming Considerations," defines the operands and possible values for each.) **Instruction** **Quotient Stored in:** **V Flag Set if Quotient is:** DIVB Short-Integer < –128 or > +127 (< 81H or > 7FH) DIV Integer < –32768 or > +32767 (< 8001H or > 7FFFH) DIVUB Byte > 255 (FFH) DIVU Word > 65535 (FFFFH)
VT	The overflow-trap flag is set when the overflow flag is set, but it is cleared only by the CLRVT, JVT, and JNVT instructions. This allows testing for a possible overflow at the end of a sequence of related arithmetic operations, which is generally more efficient than testing the overflow flag after each operation.
Z	The zero flag is set to indicate that the result of an operation was zero. For multiple-precision calculations, the zero flag cannot be set by the instructions that use the carry bit from the previous calculation (e.g., ADDC, SUBC). However, these instructions can clear the zero flag. This ensures that the zero flag will reflect the result of the entire operation, not just the last calculation. For example, if the result of adding together the lower words of two double words is zero, the zero flag would be set. When the upper words are added together using the ADDC instruction, the flag remains set if the result is zero and is cleared if the result is not zero.

Table A-3 shows the effect of the PSW flags or a specified condition on conditional jump instructions. Table A-4 defines the symbols used in Table A-6 to show the effect of each instruction on the PSW flags.

Table A-3. Effect of PSW Flags or Specified Conditions on Conditional Jump Instructions

Instruction	Jumps to Destination if	Continues if
DJNZ	decremented byte ≠ 0	decremented byte = 0
DJNZW	decremented word ≠ 0	decremented word = 0
JBC	specified register bit = 0	specified register bit = 1
JBS	specified register bit = 1	specified register bit = 0
JNC	C = 0	C = 1
JNH	C = 0 OR Z = 1	C = 1 AND Z = 0
JC	C = 1	C = 0
JH	C = 1 AND Z = 0	C = 0 OR Z = 1
JGE	N = 0	N = 1
JGT	N = 0 AND Z = 0	N = 1 OR Z = 1
JLT	N = 1	N = 0
JLE	N = 1 OR Z = 1	N = 0 AND Z = 0
JNST	ST = 0	ST = 1
JST	ST = 1	ST = 0
JNV	V = 0	V = 1
JV	V = 1	V = 0
JNVT	VT = 0	VT = 1 (clears VT)
JVT	VT = 1 (clears VT)	VT = 0
JNE	Z = 0	Z = 1
JE	Z = 1	Z = 0

Table A-4. PSW Flag Setting Symbols

Symbol	Description
✓	The instruction sets or clears the flag, as appropriate.
—	The instruction does not modify the flag.
↓	The instruction may clear the flag, if it is appropriate, but cannot set it.
↑	The instruction may set the flag, if it is appropriate, but cannot clear it.
1	The instruction sets the flag.
0	The instruction clears the flag.
?	The instruction leaves the flag in an indeterminate state.

Table A-5 defines the variables that are used in Table A-6 to represent the instruction operands.

Table A-5. Operand Variables

Variable	Description
aa	A 2-bit field within an opcode that selects the basic addressing mode used. This field is present only in those opcodes that allow addressing mode options. The field is encoded as follows: 00 register-direct 01 immediate 10 indirect 11 indexed
baop	A byte operand that is addressed by any addressing mode.
bbb	A 3-bit field within an opcode that selects a specific bit within a register.
bitno	A 3-bit field within an opcode that selects one of the eight bits in a byte.
breg	A byte register in the internal register file. When it could be unclear whether this variable refers to a source or a destination register, it is prefixed with an *S* or a *D*. The value must be in the range of 00–FFH.
cadd	An address in the program code.
Dbreg†	A byte register in the lower register file that serves as the destination of the instruction operation.
disp	Displacement. The distance between the end of an instruction and the target label.
Dlreg†	A 32-bit register in the lower register file that serves as the destination of the instruction operation. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
Dwreg†	A word register in the lower register file that serves as the destination of the instruction operation. Must be aligned on an address that is evenly divisible by 2. The value must be in the range of 00–FEH.
lreg	A 32-bit register in the lower register file. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
ptr2_reg	A double-pointer register, used with the EBMOVI instruction. Must be aligned on an address that is evenly divisible by 8. The value must be in the range of 00–F8H.
preg	A pointer register. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
Sbreg†	A byte register in the lower register file that serves as the source of the instruction operation.
Slreg†	A 32-bit register in the lower register file that serves as the source of the instruction operation. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
Swreg†	A word register in the lower register file that serves as the source of the instruction operation. Must be aligned on an address that is evenly divisible by 2. The value must be in the range of 00–FEH.
treg	A 24-bit register in the lower register file. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH.
waop	A word operand that is addressed by any addressing mode.
w2_reg	A double-word register in the lower register file. Must be aligned on an address that is evenly divisible by 4. The value must be in the range of 00–FCH. Although *w2_reg* is similar to *lreg*, there is a distinction: *w2_reg* consists of two halves, each containing a 16-bit address; *lreg* is indivisible and contains a 32-bit number.
wreg	A word register in the lower register file. When it could be unclear whether this variable refers to a source or a destination register, it is prefixed with an *S* or a *D*. Must be aligned on an address that is evenly divisible by 2. The value must be in the range of 00–FEH.
xxx	The three high-order bits of displacement.

† The *D* or *S* prefix is used only when it could be unclear whether a variable refers to a destination or a source register.

Table A-6. Instruction Set

Mnemonic	Operation	Instruction Format
ADD (2 operands)	ADD WORDS. Adds the source and destination word operands and stores the sum into the destination operand. (DEST) ← (DEST) + (SRC) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC ADD wreg, waop (011001aa) (waop) (wreg)
ADD (3 operands)	ADD WORDS. Adds the two source word operands and stores the sum into the destination operand. (DEST) ← (SRC1) + (SRC2) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC1, SRC2 ADD Dwreg, Swreg, waop (010001aa) (waop) (Swreg) (Dwreg)
ADDB (2 operands)	ADD BYTES. Adds the source and destination byte operands and stores the sum into the destination operand. (DEST) ← (DEST) + (SRC) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC ADDB breg, baop (011101aa) (baop) (breg)
ADDB (3 operands)	ADD BYTES. Adds the two source byte operands and stores the sum into the destination operand. (DEST) ← (SRC1) + (SRC2) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC1, SRC2 ADDB Dbreg, Sbreg, baop (010101aa) (baop) (Sbreg) (Dbreg)
ADDC	ADD WORDS WITH CARRY. Adds the source and destination word operands and the carry flag (0 or 1) and stores the sum into the destination operand. (DEST) ← (DEST) + (SRC) + C **PSW Flag Settings** Z: ↓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC ADDC wreg, waop (101001aa) (waop) (wreg)

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ADDCB</td><td>ADD BYTES WITH CARRY. Adds the source and destination byte operands and the carry flag (0 or 1) and stores the sum into the destination operand.
(DEST) ← (DEST) + (SRC) + C
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>↓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
ADDCB breg, baop
(101101aa) (baop) (breg)</td></tr>
<tr><td>AND
(2 operands)</td><td>LOGICAL AND WORDS. ANDs the source and destination word operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) AND (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
AND wreg, waop
(011000aa) (waop) (wreg)</td></tr>
<tr><td>AND
(3 operands)</td><td>LOGICAL AND WORDS. ANDs the two source word operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (SRC1) AND (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC1, SRC2
AND Dwreg, Swreg, waop
(010000aa) (waop) (Swreg) (Dwreg)</td></tr>
<tr><td>ANDB
(2 operands)</td><td>LOGICAL AND BYTES. ANDs the source and destination byte operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) AND (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ANDB breg, baop
(011100aa) (baop) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ANDB
(3 operands)</td><td>LOGICAL AND BYTES. ANDs the two source byte operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a “1” and zeros in all other bit positions.
(DEST) ← (SRC1) AND (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC1, SRC2
ANDB Dbreg, Sbreg, baop
(010100aa) (baop) (Sbreg) (Dbreg)</td></tr>
<tr><td>BMOV</td><td>BLOCK MOVE. Moves a block of word data from one location in memory to another. The source and destination addresses are calculated using the indirect with autoincrement addressing mode. A long register (PTRS) addresses the source and destination pointers, which are stored in adjacent word registers. The source pointer (SRCPTR) is the low word and the destination pointer (DSTPTR) is the high word of PTRS. A word register (CNTREG) specifies the number of transfers. The blocks of data can be located anywhere in page 00H of register RAM, but should not overlap. Because the source (SRCPTR) and destination (DSTPTR) pointers are 16 bits wide, this instruction uses nonextended data moves. It cannot operate across page boundaries. For example, SRCPTR cannot point to a location on page 05 while DSTPTR points to page 00. SRCPTR and DSTPTR will operate from the page defined by EP_REG. EP_REG should be set to 00H to select page 00H (see “Accessing Data” on page 5-23). (The 80C196NU forces EP_REG to 00H.)
COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS + 2) ← DSTPTR + 2
COUNT ← COUNT – 1
if COUNT ≠ 0 then
go to LOOP
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PTRS, CNTREG
BMOV lreg, wreg
(11000001) (wreg) (lreg)

NOTE: The pointers are autoincremented during this instruction. However, CNTREG is not decremented. Therefore, it is easy to unintentionally create a long, uninterruptible operation with the BMOV instruction. Use the BMOVI instruction for an interruptible operation.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>BMOVI</td><td>INTERRUPTIBLE BLOCK MOVE. Moves a block of word data from one location in memory to another. The instruction is identical to BMOV, except that BMOVI is interruptible. The source and destination addresses are calculated using the indirect with autoincrement addressing mode. A long register (PTRS) addresses the source and destination pointers, which are stored in adjacent word registers. The source pointer (SRCPTR) is the low word and the destination pointer (DSTPTR) is the high word of PTRS. A word register (CNTREG) specifies the number of transfers. The blocks of data can be located anywhere in page 00H of register RAM, but should not overlap. Because the source (SRCPTR) and destination (DSTPTR) pointers are 16 bits wide, this instruction uses nonexteneded data moves. It cannot operate across page boundaries. (If you need to cross page boundaries, use the EBMOVI instruction.) PTSSRC and PTSDST will operate from the page defined by EP_REG. EP_REG should be set to 00H to select page 00H (see “Accessing Data” on page 5-23). (The 80C196NU forces EP_REG to 00H.)
COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS + 2) ← DSTPTR + 2
COUNT ← COUNT – 1
if COUNT ≠ 0 then
go to LOOP
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PTRS, CNTREG
BMOVI lreg, wreg
(11001101) (wreg) (lreg)

NOTE: The pointers are autoincremented during this instruction. However, CNTREG is decremented only when the instruction is interrupted. When BMOVI is interrupted, CNTREG is updated to store the interim word count at the time of the interrupt. For this reason, you should always reload CNTREG before starting a BMOVI.</td></tr>
<tr><td>BR</td><td>BRANCH INDIRECT. Continues execution at the address specified in the operand word register.
PC ← (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST
BR [wreg]
(11100011) (wreg)

NOTE: In 1-Mbyte mode, the BR instruction always branches to page FFH. Use the EBR instruction to branch to an address on any other page.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>CLR</td><td>CLEAR WORD. Clears the value of the operand.
(DEST) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST
CLR wreg
(00000001) (wreg)</td></tr>
<tr><td>CLRB</td><td>CLEAR BYTE. Clears the value of the operand.
(DEST) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST
CLRB breg
(00010001) (breg)</td></tr>
<tr><td>CLRC</td><td>CLEAR CARRY FLAG. Clears the carry flag.
C ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>0</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>CLRC
(11111000)</td></tr>
<tr><td>CLRVT</td><td>CLEAR OVERFLOW-TRAP FLAG. Clears the overflow-trap flag.
VT ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>—</td></tr></table></td><td>CLRVT
(11111100)</td></tr>
<tr><td>CMP</td><td>COMPARE WORDS. Subtracts the source word operand from the destination word operand. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.
(DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
CMP wreg, waop
(100010aa) (waop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>CMPB</td><td>COMPARE BYTES. Subtracts the source byte operand from the destination byte operand. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.
(DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
CMPB breg, baop
(100110aa) (baop) (breg)</td></tr>
<tr><td>CMPL</td><td>COMPARE LONG. Compares the magnitudes of two double-word (long) operands. The operands are specified using the direct addressing mode. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.
(DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>—</td></tr></table></td><td>DEST, SRC
CMPL Dlreg, Slreg
(11000101) (Slreg) (Dlreg)</td></tr>
<tr><td>DEC</td><td>DECREMENT WORD. Decrements the value of the operand by one.
(DEST) ← (DEST) –1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST
DEC wreg
(00000101) (wreg)</td></tr>
<tr><td>DECB</td><td>DECREMENT BYTE. Decrements the value of the operand by one.
(DEST) ← (DEST) –1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST
DECB breg
(00010101) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>DI</td><td>DISABLE INTERRUPTS. Disables interrupts. Interrupt calls cannot occur after this instruction.
Interrupt Enable (PSW.1) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DI
(11111010)</td></tr>
<tr><td>DIV</td><td>DIVIDE INTEGERS. Divides the contents of the destination long-integer operand by the contents of the source integer word operand, using signed arithmetic. It stores the quotient into the low-order word of the destination (i.e., the word with the lower address) and the remainder into the high-order word. The following two statements are performed concurrently.
(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
DIV lreg, waop
(11111110) (100011aa) (waop) (lreg)</td></tr>
<tr><td>DIVB</td><td>DIVIDE SHORT-INTEGERS. Divides the contents of the destination integer operand by the contents of the source short-integer operand, using signed arithmetic. It stores the quotient into the low-order byte of the destination (i.e., the word with the lower address) and the remainder into the high-order byte. The following two statements are performed concurrently.
(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
DIVB wreg, baop
(11111110) (100111aa) (baop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>DIVU</td><td>DIVIDE WORDS, UNSIGNED. Divides the contents of the destination double-word operand by the contents of the source word operand, using unsigned arithmetic. It stores the quotient into the low-order word (i.e., the word with the lower address) of the destination operand and the remainder into the high-order word. The following two statements are performed concurrently.
(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
DIVU lreg, waop
(100011aa) (waop) (lreg)</td></tr>
<tr><td>DIVUB</td><td>DIVIDE BYTES, UNSIGNED. This instruction divides the contents of the destination word operand by the contents of the source byte operand, using unsigned arithmetic. It stores the quotient into the low-order byte (i.e., the byte with the lower address) of the destination operand and the remainder into the high-order byte. The following two statements are performed concurrently.
(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
DIVUB wreg, baop
(100111aa) (baop) (wreg)</td></tr>
<tr><td>DJNZ</td><td>DECREMENT AND JUMP IF NOT ZERO. Decrements the value of the byte operand by 1. If the result is 0, control passes to the next sequential instruction. If the result is not 0, the instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
(COUNT) ← (COUNT) –1
if (COUNT) ≠ 0 then
PC ← PC + 8-bit disp
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DJNZ breg,cadd
(11100000) (breg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>DJNZW</td><td>DECREMENT AND JUMP IF NOT ZERO WORD. Decrements the value of the word operand by 1. If the result is 0, control passes to the next sequential instruction. If the result is not 0, the instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127
(COUNT) ← (COUNT) –1
if (COUNT) ≠ 0 then
PC ← PC + 8-bit disp
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DJNZW wreg,cadd
(11100001) (wreg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits</td></tr>
<tr><td>DPTS</td><td>DISABLE PERIPHERAL TRANSACTION SERVER (PTS). Disables the peripheral transaction server (PTS).
PTS Disable (PSW.2) ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DPTS
(11101100)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>EBMOVI</td><td>EXTENDED INTERRUPTABLE BLOCK MOVE. Moves a block of word data from one memory location to another. This instruction allows you to move blocks of up to 64K words between any two locations in the 16-Mbyte address space. This instruction is interruptable.

The source and destination addresses are calculated using the extended indirect with autoincrement addressing mode. A quad-word register (PTRS) addresses the 24-bit source and destination pointers, which are stored in adjacent double-word registers. The source pointer (SRCPTR) is the low double-word and the destination pointer is the high double-word of PTRS. A word register (CNTREG) specifies the number of transfers. The blocks of data can reside anywhere in memory, but should not overlap.

COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS + 2) ← DSTPTR + 2
COUNT ← COUNT 1
if COUNT ≠ 0 then
go to LOOP

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PTRS, CNTREG
EBMOVI prt2_reg, wreg
(11100100) (wreg) (prt2_reg)

NOTES: The pointers are autoincremented during this instruction. However, CNTREG is decremented only when the instruction is interrupted. When EBMOVI is interrupted, CNTREG is updated to store the interim word count at the time of the interrupt. For this reason, you should always reload CNTREG before starting an EBMOVI.

For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
<tr><td>EBR</td><td>EXTENDED BRANCH INDIRECT. Continues execution at the address specified in the operand word register. This instruction is an unconditional indirect jump to anywhere in the 16-Mbyte address space.

EBR shares its opcode (E3) with the BR instruction. To differentiate between the two, the compiler sets the least-significant bit of the EBR instruction. For example: EBR [50] becomes E351 when compiled.

PC ← (DEST)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST
EBR cadd
or
EBR [treg]
(11100011) (treg)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ECALL</td><td>EXTENDED CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The operand may be any address in the address space.

This instruction is an unconditional relative call to anywhere in the 16-Mbyte address space. It functions only in extended addressing mode.

SP ← SP – 4
(SP) ← PC
PC ← PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>ECALL cadd
(1111 0001) (disp-low) (disp-high) (disp-ext)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
<tr><td>EI</td><td>ENABLE INTERRUPTS. Enables interrupts following the execution of the next statement. Interrupt calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.1) ← 1

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>EI
(11111011)</td></tr>
<tr><td>EJMP</td><td>EXTENDED JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The operand may be any address in the entire address space. The offset must be in the range of +8,388,607 to –8,388,608 for 24-bit addresses.

This instruction is an unconditional, relative jump to anywhere in the 16-Mbyte address space. It functions only in extended addressing mode.

PC ←PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>EJMP cadd
(11100110) (disp-low) (disp-high) (disp-ext)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ELD</td><td>EXTENDED LOAD WORD. Loads the value of the source word operand into the destination operand.
This instruction allows you to move data from anywhere in the 16-Mbyte address space into the lower register file.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ELD wreg, [treg]
ext. indirect: (11101000) (treg) (wreg)
ext. indexed: (11101001) (treg) (disp-low) (disp-high) (disp-ext) (wreg)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
<tr><td>ELDB</td><td>EXTENDED LOAD BYTE. Loads the value of the source byte operand into the destination operand.
This instruction allows you to move data from anywhere in the 16-Mbyte address space into the lower register file.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ELDB breg, [treg]
ext. indirect: (11101010) (treg) (breg)
ext. indexed: (11101011) (treg) (disp-low) (disp-high) (disp-ext) (breg)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
<tr><td>EPTS</td><td>ENABLE PERIPHERAL TRANSACTION SERVER (PTS). Enables the peripheral transaction server (PTS).
PTS Enable (PSW.2) ← 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>EPTS
(11101101)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>EST</td><td>EXTENDED STORE WORD. Stores the value of the source (leftmost) word operand into the destination (rightmost) operand.
This instruction allows you to move data from the lower register file to anywhere in the 16-Mbyte address space.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
EST wreg, [treg]
ext. indirect: (00011100) (treg) (wreg)
ext. indexed: (00011101) (treg) (disp-low) (disp-high) (disp-ext) (wreg)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
<tr><td>ESTB</td><td>EXTENDED STORE BYTE. Stores the value of the source (leftmost) byte operand into the destination (rightmost) operand.
This instruction allows you to move data from the lower register file to anywhere in the 16-Mbyte address space.
ext. indirect: (DEST) ← (SRC)
ext indexed: (DEST)← (SRC) + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
ESTB breg, [treg]
ext. indirect: (00011110) (treg) (breg)
ext. indexed: (00011111) (treg) (disp-low) (disp-high) (disp-ext) (breg)

NOTE: For 20-bit addresses, the offset must be in the range of +524287 to –524288.</td></tr>
<tr><td>EXT</td><td>SIGN-EXTEND INTEGER INTO LONG-INTEGER. Sign-extends the low-order word of the operand throughout the high-order word of the operand.
if DEST.15 = 1 then
(high word DEST) ← 0FFFFH
else
(high word DEST) ← 0
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>EXT lreg
(00000110) (lreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>EXTB</td><td>SIGN-EXTEND SHORT-INTEGER INTO INTEGER. Sign-extends the low-order byte of the operand throughout the high-order byte of the operand.
if DEST.7 = 1 then
(high byte DEST) ← 0FFH
else
(high byte DEST) ← 0
end_if
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>EXTB wreg
(00010110) (wreg)</td></tr>
<tr><td>IDLPD</td><td>IDLE/POWERDOWN. Depending on the 8-bit value of the KEY operand, this instruction causes the device
• to enter idle mode, KEY=1,
• to enter powerdown mode, KEY=2,
• to enter standby mode, KEY=3, (NU only)
• to execute a reset sequence, KEY = any value other than 1 or 2 (NP) or 1, 2, or 3 (NU).
The bus controller completes any prefetch cycle in progress before the CPU stops or resets.
if KEY = 1 then
enter idle
else if KEY = 2 then
enter powerdown
else if KEY = 3 then
enter standby (NU only)
else
execute reset
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td colspan="6">KEY = 1 or 2 (NP) or 1, 2, or 3 (NU)</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr><tr><td colspan="6">KEY = any value other than 1 or 2 (NP) or 1, 2, or 3 (NU)</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>IDLPD #key
(11110110) (key)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>INC</td><td>INCREMENT WORD. Increments the value of the word operand by 1.
(DEST) ← (DEST) + 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>0</td></tr></table></td><td>INC wreg
(00000111) (wreg)</td></tr>
<tr><td>INCB</td><td>INCREMENT BYTE. Increments the value of the byte operand by 1.
(DEST) ← (DEST) + 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>INCB breg
(00010111) (breg)</td></tr>
<tr><td>JBC</td><td>JUMP IF BIT IS CLEAR. Tests the specified bit. If the bit is set, control passes to the next sequential instruction. If the bit is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if (specified bit) = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JBC breg,bitno,cadd
(00110bbb) (breg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JBS</td><td>JUMP IF BIT IS SET. Tests the specified bit. If the bit is clear, control passes to the next sequential instruction. If the bit is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if (specified bit) = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JBS breg,bitno,cadd
(00111bbb) (breg) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JC</td><td>JUMP IF CARRY FLAG IS SET. Tests the carry flag. If the carry flag is clear, control passes to the next sequential instruction. If the carry flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if C = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JC cadd
(11011011) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JE</td><td>JUMP IF EQUAL. Tests the zero flag. If the flag is clear, control passes to the next sequential instruction. If the zero flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if Z = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JE cadd
(11011111) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JGE</td><td>JUMP IF SIGNED GREATER THAN OR EQUAL. Tests the negative flag. If the negative flag is set, control passes to the next sequential instruction. If the negative flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JGE cadd
(11010110) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JGT</td><td>JUMP IF SIGNED GREATER THAN. Tests both the zero flag and the negative flag. If either flag is set, control passes to the next sequential instruction. If both flags are clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 0 AND Z = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JGT cadd
(11010010) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JH</td><td>JUMP IF HIGHER (UNSIGNED). Tests both the zero flag and the carry flag. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction. If the carry flag is set and the zero flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if C = 1 AND Z = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JH cadd
(11011001) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JLE</td><td>JUMP IF SIGNED LESS THAN OR EQUAL. Tests both the negative flag and the zero flag. If both flags are clear, control passes to the next sequential instruction. If either flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 1 OR Z = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JLE cadd
(11011010) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JLT</td><td>JUMP IF SIGNED LESS THAN. Tests the negative flag. If the flag is clear, control passes to the next sequential instruction. If the negative flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if N = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JLT cadd
(11011110) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNC</td><td>JUMP IF CARRY FLAG IS CLEAR. Tests the carry flag. If the flag is set, control passes to the next sequential instruction. If the carry flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if C = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNC cadd
(11010011) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNE</td><td>JUMP IF NOT EQUAL. Tests the zero flag. If the flag is set, control passes to the next sequential instruction. If the zero flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.
if Z = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNE cadd
(11010111) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JNH</td><td>JUMP IF NOT HIGHER (UNSIGNED). Tests both the zero flag and the carry flag. If the carry flag is set and the zero flag is clear, control passes to the next sequential instruction. If either the carry flag is clear or the zero flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if C = 0 OR Z = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNH cadd
(11010001) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNST</td><td>JUMP IF STICKY BIT FLAG IS CLEAR. Tests the sticky bit flag. If the flag is set, control passes to the next sequential instruction. If the sticky bit flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if ST = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNST cadd
(11010000) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JNV</td><td>JUMP IF OVERFLOW FLAG IS CLEAR. Tests the overflow flag. If the flag is set, control passes to the next sequential instruction. If the overflow flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if V = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JNV cadd
(11010101) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JNVT</td><td>JUMP IF OVERFLOW-TRAP FLAG IS CLEAR. Tests the overflow-trap flag. If the flag is set, this instruction clears the flag and passes control to the next sequential instruction. If the overflow-trap flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if VT = 0 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>—</td></tr></table></td><td>JNVT cadd
(11010100) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JST</td><td>JUMP IF STICKY BIT FLAG IS SET. Tests the sticky bit flag. If the flag is clear, control passes to the next sequential instruction. If the sticky bit flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if ST = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JST cadd
(11011000) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>JV</td><td>JUMP IF OVERFLOW FLAG IS SET. Tests the overflow flag. If the flag is clear, control passes to the next sequential instruction. If the overflow flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if V = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>JV cadd
(11011101) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>JVT</td><td>JUMP IF OVERFLOW-TRAP FLAG IS SET. Tests the overflow-trap flag. If the flag is clear, control passes to the next sequential instruction. If the overflow-trap flag is set, this instruction clears the flag and adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.
if VT = 1 then
PC ← PC + 8-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>—</td></tr></table></td><td>JVT cadd
(11011100) (disp)

NOTE: The displacement (disp) is sign-extended to 24 bits.</td></tr>
<tr><td>LCALL</td><td>LONG CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The offset must be in the range of –32,768 to +32,767.
64-Kbyte mode:
SP ← SP – 2
(SP) ← PC
PC ← PC + 16-bit disp
1-Mbyte mode:
SP ← SP – 4
(SP) ← PC
PC ← PC + 24-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>LCALL cadd
(11101111) (disp-low) (disp-high)

NOTE: The displacement (disp) is sign-extended to 24 bits in the 1-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary.</td></tr>
<tr><td>LD</td><td>LOAD WORD. Loads the value of the source word operand into the destination operand.
(DEST) ← (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LD wreg, waop
(101000aa) (waop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>LDB</td><td>LOAD BYTE. Loads the value of the source byte operand into the destination operand.
(DEST) ← (SRC)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LDB breg, baop
(101100aa) (baop) (breg)</td></tr>
<tr><td>LDBSE</td><td>LOAD BYTE SIGN-EXTENDED. Sign-extends the value of the source short-integer operand and loads it into the destination integer operand.
(low byte DEST) ← (SRC)
if DEST.15 = 1 then
(high word DEST) ← 0FFH
else
(high word DEST) ← 0
end_if

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LDBSE wreg, baop
(101111aa) (baop) (wreg)</td></tr>
<tr><td>LDBZE</td><td>LOAD BYTE ZERO-EXTENDED. Zero-extends the value of the source byte operand and loads it into the destination word operand.
(low byte DEST) ← (SRC)
(high byte DEST) ← 0

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
LDBZE wreg, baop
(101011aa) (baop) (wreg)</td></tr>
<tr><td>LJMP</td><td>LONG JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –32,768 to +32,767.
64-Kbyte mode:
PC ← PC + 16-bit disp
1-Mbyte mode:
PC ← PC + 24-bit disp

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>LJMP cadd
(11100111) (disp-low) (disp-high)

NOTE: The displacement (disp) is sign-extended to 24 bits in the 1-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>MUL
(2 operands)</td><td>MULTIPLY INTEGERS. Multiplies the source and destination integer operands, using signed arithmetic, and stores the 32-bit result into the destination long-integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MUL lreg, waop
(11111110) (011011aa) (waop) (lreg)</td></tr>
<tr><td>MUL
(3 operands)</td><td>MULTIPLY INTEGERS. Multiplies the two source integer operands, using signed arithmetic, and stores the 32-bit result into the destination long-integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MUL lreg, wreg, waop
(11111110) (010011aa) (waop) (wreg) (lreg)

NOTE: (8XC196NU only.) A destination address in the range 00H–0FH enables the multiply-accumulate function. When set, bit 3 of the destination address causes the accumulator to be cleared before the results of the multiply are added to the contents of the accumulator. For example, if the destination address is 08H, the accumulator is cleared and then the results of the multiply are added. However, if the destination address is 00H, the results of the multiply are added to the current contents of the accumulator.</td></tr>
<tr><td>MULB
(2 operands)</td><td>MULTIPLY SHORT-INTEGERS. Multiplies the source and destination short-integer operands, using signed arithmetic, and stores the 16-bit result into the destination integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MULB wreg, baop
(11111110) (011111aa) (baop) (wreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>MULB
(3 operands)</td><td>MULTIPLY SHORT-INTEGERS. Multiplies the two source short-integer operands, using signed arithmetic, and stores the 16-bit result into the destination integer operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MULB wreg, breg, baop
(11111110) (010111aa) (baop) (breg) (wreg)</td></tr>
<tr><td>MULU
(2 operands)</td><td>MULTIPLY WORDS, UNSIGNED. Multiplies the source and destination word operands, using unsigned arithmetic, and stores the 32-bit result into the destination double-word operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MULU lreg, waop
(011011aa) (waop) (lreg)</td></tr>
<tr><td>MULU
(3 operands)</td><td>MULTIPLY WORDS, UNSIGNED. Multiplies the two source word operands, using unsigned arithmetic, and stores the 32-bit result into the destination double-word operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MULU lreg, wreg, waop
(010011aa) (waop) (wreg) (lreg)

NOTE: (8XC196NU only.) A destination address in the range 00H–0FH enables the multiply-accumulate function. When set, bit 3 of the destination address causes the accumulator to be cleared before the results of the multiply are added to the contents of the accumulator. For example, if the destination address is 08H, the accumulator is cleared and then the results of the multiply are added. However, if the destination address is 00H, the results of the multiply are added to the current contents of the accumulator.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>MULUB
(2 operands)</td><td>MULTIPLY BYTES, UNSIGNED. Multiplies the source and destination operands, using unsigned arithmetic, and stores the word result into the destination operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (DEST) × (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC
MULUB wreg, baop
(011111aa) (baop) (wreg)</td></tr>
<tr><td>MULUB
(3 operands)</td><td>MULTIPLY BYTES, UNSIGNED. Multiplies the two source byte operands, using unsigned arithmetic, and stores the word result into the destination operand. The sticky bit flag is undefined after the instruction is executed.
(DEST) ← (SRC1) × (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>?</td></tr></table></td><td>DEST, SRC1, SRC2
MULUB wreg, breg, baop
(010111aa) (baop) (breg) (wreg)</td></tr>
<tr><td>NEG</td><td>NEGATE INTEGER. Negates the value of the integer operand.
(DEST) ← – (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>NEG wreg
(00000011) (wreg)</td></tr>
<tr><td>NEGB</td><td>NEGATE SHORT-INTEGER. Negates the value of the short-integer operand.
(DEST) ← – (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>NEGB breg
(00010011) (breg)</td></tr>
<tr><td>NOP</td><td>NO OPERATION. Does nothing. Control passes to the next sequential instruction.
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>NOP
(11111101)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>NORML</td><td>NORMALIZE LONG-INTEGER. Normalizes the source (leftmost) long-integer operand. (That is, it shifts the operand to the left until its most significant bit is “1” or until it has performed 31 shifts). If the most significant bit is still “0” after 31 shifts, the instruction stops the process and sets the zero flag. The instruction stores the actual number of shifts performed in the destination (rightmost) operand.
(COUNT) ← 0
do while
(MSB (DEST) = 0) AND (COUNT) < 31)
(DEST) ← (DEST) × 2
(COUNT) ← (COUNT) + 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>?</td><td>0</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
NORML lreg, breg
(00001111) (breg) (lreg)</td></tr>
<tr><td>NOT</td><td>COMPLEMENT WORD. Complements the value of the word operand (replaces each “1” with a “0” and each “0” with a “1”).
(DEST) ← NOT (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>NOT wreg
(00000010) (wreg)</td></tr>
<tr><td>NOTB</td><td>COMPLEMENT BYTE. Complements the value of the byte operand (replaces each “1” with a “0” and each “0” with a “1”).
(DEST) ← NOT (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>NOTB breg
(00010010) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>OR</td><td>LOGICAL OR WORDS. ORs the source word operand with the destination word operand and replaces the original destination operand with the result. The result has a “1” in each bit position in which either the source or destination operand had a “1”.
(DEST) ← (DEST) OR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
OR wreg, waop
(100000aa) (waop) (wreg)</td></tr>
<tr><td>ORB</td><td>LOGICAL OR BYTES. ORs the source byte operand with the destination byte operand and replaces the original destination operand with the result. The result has a “1” in each bit position in which either the source or destination operand had a “1”.
(DEST) ← (DEST) OR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
ORB breg, baop
(100100aa) (baop) (breg)</td></tr>
<tr><td>POP</td><td>POP WORD. Pops the word on top of the stack and places it at the destination operand.
(DEST) ← (SP)
SP ← SP + 2
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>POP waop
(110011aa) (waop)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>POPA</td><td>POP ALL. This instruction is used instead of POPF, to support the eight additional interrupts. It pops two words off the stack and places the first word into the INT_MASK1/WSR register pair and the second word into the PSW/INT_MASK register-pair. This instruction increments the SP by 4. Interrupt calls cannot occur immediately following this instruction.
INT_MASK1/WSR ← (SP)
SP ← SP + 2
PSW/INT_MASK ← (SP)
SP ← SP + 2
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr></table></td><td>POPA
(11110101)</td></tr>
<tr><td>POPF</td><td>POP FLAGS. Pops the word on top of the stack and places it into the PSW. Interrupt calls cannot occur immediately following this instruction.
(PSW) ← (SP)
SP ← SP + 2
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr></table></td><td>POPF
(11110011)</td></tr>
<tr><td>PUSH</td><td>PUSH WORD. Pushes the word operand onto the stack.
SP ← SP – 2
(SP) ← (DEST)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>PUSH waop
(110010aa) (waop)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>PUSHA</td><td>PUSH ALL. This instruction is used instead of PUSHF, to support the eight additional interrupts. It pushes two words — PSW/INT_MASK and INT_MASK1/WSR — onto the stack.
This instruction clears the PSW, INT_MASK, and INT_MASK1 registers and decrements the SP by 4. Interrupt calls cannot occur immediately following this instruction.
SP ← SP – 2
(SP) ← PSW/INT_MASK
PSW/INT_MASK ← 0
SP ← SP – 2
(SP) ← INT_MASK1/WSR
INT_MASK1 ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>PUSHA
(11110100)</td></tr>
<tr><td>PUSHF</td><td>PUSH FLAGS. Pushes the PSW onto the top of the stack, then clears it. Clearing the PSW disables interrupt servicing. Interrupt calls cannot occur immediately following this instruction.
SP ← SP – 2
(SP) ← PSW/INT_MASK
PSW/INT_MASK ← 0
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>PUSHF
(11110010)</td></tr>
<tr><td>RET</td><td>RETURN FROM SUBROUTINE. Pops the PC off the top of the stack.
64-Kbyte mode:
PC ← (SP)
SP ← SP + 2
1-Mbyte mode:
PC ← (SP)
SP ← SP + 4
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>RET
(11110000)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>RST</td><td>RESET SYSTEM. Initializes the PSW to zero, the EPC/PC to FF2080H, and the pins and SFRs to their reset values. Executing this instruction causes the RESET# pin to be pulled low for 16 state times.
SFR ← Reset Status
Pin ← Reset Status
PSW ← 0
EPC/PC ← FF2080H
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>RST
(11111111)</td></tr>
<tr><td>SCALL</td><td>SHORT CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The offset must be in the range of –1024 to +1023.
64-Kbyte mode:
SP ← SP – 2
(SP) ← PC
PC←PC+11-bit disp
1-Mbyte mode:
SP ← SP – 4
(SP) ← PC
PC←PC+11-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SCALL cadd
(00101xxx) (disp-low)
NOTE: The displacement (disp) is sign-extended to 16-bits in the 64-Kbyte addressing mode and to 24 bits in the 1-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary in 1-Mbyte mode.</td></tr>
<tr><td>SETC</td><td>SET CARRY FLAG. Sets the carry flag.
C ← 1
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>1</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SETC
(11111001)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHL</td><td>SHIFT WORD LEFT. Shifts the destination word operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← High order bit of (DEST)
(DEST) ← (DEST) × 2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>SHL wreg,#count
(00001001) (count) (wreg)
or
SHL wreg,breg
(00001001) (breg) (wreg)</td></tr>
<tr><td>SHLB</td><td>SHIFT BYTE LEFT. Shifts the destination byte operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← High order bit of (DEST)
(DEST) ← (DEST) × 2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>SHLB breg,#count
(00011001) (count) (breg)
or
SHLB breg,breg
(00011001) (breg) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHLL</td><td>SHIFT DOUBLE-WORD LEFT. Shifts the destination double-word operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← High order bit of (DEST)
(DEST) ← (DEST) × 2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>SHLL lreg,#count
(00001101) (count) (breg)
or
SHLL lreg,breg
(00001101) (breg) (lreg)</td></tr>
<tr><td>SHR</td><td>LOGICAL RIGHT SHIFT WORD. Shifts the destination word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>0</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHR wreg,#count
(00001000) (count) (wreg)
or
SHR wreg,breg
(00001000) (breg) (wreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, DEST/2 represents unsigned division.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHRA</td><td>ARITHMETIC RIGHT SHIFT WORD. Shifts the destination word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was “0,” zeros are shifted in. If the value was “1,” ones are shifted in. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRA wreg,#count
(00001010) (count) (wreg)
or
SHRA wreg,breg
(00001010) (breg) (wreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, DEST/2 represents signed division.</td></tr>
<tr><td>SHRAB</td><td>ARITHMETIC RIGHT SHIFT BYTE. Shifts the destination byte operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was “0,” zeros are shifted in. If the value was “1,” ones are shifted in. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C = Low order bit of (DEST)
(DEST) ← (DEST)/2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRAB breg,#count
(00011010) (count) (breg)
or
SHRAB breg,breg
(00011010) (breg) (breg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, DEST/2 represents signed division.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHRAL</td><td>ARITHMETIC RIGHT SHIFT DOUBLE-WORD. Shifts the destination double-word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was “0,” zeros are shifted in. If the value was “1,” ones are shifted in.
Temp ← (COUNT)
do while Temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRAL lreg,#count
(00001110) (count) (lreg)
or
SHRAL lreg,breg
(00001110) (breg) (lreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, DEST/2 represents signed division.</td></tr>
<tr><td>SHRB</td><td>LOGICAL RIGHT SHIFT BYTE. Shifts the destination byte operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
Temp ← Temp–1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>0</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRB breg,#count
(00011000) (count) (breg)
or
SHRB breg,breg
(00011000) (breg) (breg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, DEST/2 represents unsigned division.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>SHRL</td><td>LOGICAL RIGHT SHIFT DOUBLE-WORD. Shifts the destination double-word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register (10H – 0FFH) with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeros. The last bit shifted out is saved in the carry flag.
Temp ← (COUNT)
do while Temp ≠ 0
C ← Low order bit of (DEST)
(DEST) ← (DEST)/2
Temp ← Temp – 1
end_while
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>0</td><td>✓</td><td>0</td><td>—</td><td>✓</td></tr></table></td><td>SHRL lreg,#count
(00001100) (count) (lreg)
or
SHRL lreg,breg
(00001100) (breg) (lreg)

NOTES: This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a “1” is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.

In this operation, DEST/2 represents unsigned division.</td></tr>
<tr><td>SJMP</td><td>SHORT JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –1024 to +1023, inclusive.
PC ← PC + 11-bit disp
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SJMP cadd
(00100xxx) (disp-low)

NOTE: The displacement (disp) is sign-extended to 16 bits in the 64-Kbyte addressing mode and to 24 bits in the 1-Mbyte addressing mode. This displacement may cause the program counter to cross a page boundary in 1-Mbyte mode.</td></tr>
<tr><td>SKIP</td><td>TWO BYTE NO-OPERATION. Does nothing. Control passes to the next sequential instruction. This is actually a two-byte NOP in which the second byte can be any value and is simply ignored.
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SKIP breg
(00000000) (breg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>ST</td><td>STORE WORD. Stores the value of the source (leftmost) word operand into the destination (rightmost) operand.
(DEST) ← (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
ST wreg, waop
(110000aa) (waop) (wreg)</td></tr>
<tr><td>STB</td><td>STORE BYTE. Stores the value of the source (leftmost) byte operand into the destination (rightmost) operand.
(DEST) ← (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>SRC, DEST
STB breg, baop
(110001aa) (baop) (breg)</td></tr>
<tr><td>SUB
(2 operands)</td><td>SUBTRACT WORDS. Subtracts the source word operand from the destination word operand, stores the result in the destination operand, and sets the carry flag as the complement of borrow.
(DEST) ← (DEST) – (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC
SUB wreg, waop
(011010aa) (waop) (wreg)</td></tr>
<tr><td>SUB
(3 operands)</td><td>SUBTRACT WORDS. Subtracts the first source word operand from the second, stores the result in the destination operand, and sets the carry flag as the complement of borrow.
(DEST) ← (SRC1) – (SRC2)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>↑</td><td>—</td></tr></table></td><td>DEST, SRC1, SRC2
SUB Dwreg, Swreg, waop
(010010aa) (waop) (Swreg) (Dwreg)</td></tr>
</table>

Table A-6. Instruction Set (Continued)

Mnemonic	Operation	Instruction Format
SUBB (2 operands)	SUBTRACT BYTES. Subtracts the source byte operand from the destination byte operand, stores the result in the destination operand, and sets the carry flag as the complement of borrow. (DEST) ← (DEST) – (SRC) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC SUBB breg, baop (011110aa) (baop) (breg)
SUBB (3 operands)	SUBTRACT BYTES. Subtracts the first source byte operand from the second, stores the result in the destination operand, and sets the carry flag as the complement of borrow. (DEST) ← (SRC1) – (SRC2) **PSW Flag Settings** Z: ✓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC1, SRC2 SUBB Dbreg, Sbreg, baop (010110aa) (baop) (Sbreg) (Dbreg)
SUBC	SUBTRACT WORDS WITH BORROW. Subtracts the source word operand from the destination word operand. If the carry flag was clear, SUBC subtracts 1 from the result. It stores the result in the destination operand and sets the carry flag as the complement of borrow. (DEST) ← (DEST) – (SRC) – (1–C) **PSW Flag Settings** Z: ↓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC SUBC wreg, waop (101010aa) (waop) (wreg)
SUBCB	SUBTRACT BYTES WITH BORROW. Subtracts the source byte operand from the destination byte operand. If the carry flag was clear, SUBCB subtracts 1 from the result. It stores the result in the destination operand and sets the carry flag as the complement of borrow. (DEST) ← (DEST) – (SRC) – (1–C) **PSW Flag Settings** Z: ↓ N: ✓ C: ✓ V: ✓ VT: ↑ ST: —	DEST, SRC SUBCB breg, baop (101110aa) (baop) (breg)

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>TIJMP</td><td>TABLE INDIRECT JUMP. Causes execution to continue at an address selected from a table of addresses.

The first word register, TBASE, contains the 16-bit address of the beginning of the jump table. TBASE can be located in RAM up to FEH without windowing or above FFH with windowing. The jump table itself can be placed at any nonreserved memory location on a word boundary in page FFH.

The second word register, INDEX, contains the 16-bit address that points to a register containing a 7-bit value. This value is used to calculate the offset into the jump table. Like TBASE, INDEX can be located in RAM up to FEH without windowing or above FFH with windowing. Note that the 16-bit address contained in INDEX is absolute; it disregards any windowing that may be in effect when the TIJMP instruction is executed.

The byte operand, #MASK, is 7-bit immediate data to mask INDEX. #MASK is ANDed with INDEX to determine the offset (OFFSET). OFFSET is multiplied by two, then added to the base address (TBASE) to determine the destination address (DEST X) in page FFH.

[INDEX] AND #MASK = OFFSET
(2 × OFFSET) + TBASE = DEST X
PC ← (DEST X)

<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>TIJMP TBASE, [INDEX], #MASK
(11100010) [INDEX] (#MASK) (TBASE)

NOTE: TIJMP multiplies OFFSET by two to provide for word alignment of the jump table.</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>TRAP</td><td>SOFTWARE TRAP. This instruction causes an interrupt call that is vectored through location FF2010H. The operation of this instruction is not affected by the state of the interrupt enable flag (I) in the PSW. Interrupt calls cannot occur immediately following this instruction.
64-Kbyte mode:
SP ← SP – 2
(SP) ← PC
PC ← (2010H)
1-Mbyte mode:
SP ← SP – 4
(SP) ← PC
PC ← (0FF2010H)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>TRAP
(11110111)

NOTE: This instruction is not supported by assemblers. The TRAP instruction is intended for use by development tools. These tools may not support user-application of this instruction.</td></tr>
<tr><td>XCH</td><td>EXCHANGE WORD. Exchanges the value of the source word operand with that of the destination word operand.
(DEST) ↔ (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XCH wreg, waop
(00000100) (waop) (wreg) direct
(00001011) (waop) (wreg) indexed</td></tr>
<tr><td>XCHB</td><td>EXCHANGE BYTE. Exchanges the value of the source byte operand with that of the destination byte operand.
(DEST) ↔ (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XCHB breg, baop
(00010100) (baop) (breg) direct
(00011011) (baop) (breg) indexed</td></tr>
</table>

Table A-6. Instruction Set (Continued)

<table>
<tr><th>Mnemonic</th><th>Operation</th><th>Instruction Format</th></tr>
<tr><td>XOR</td><td>LOGICAL EXCLUSIVE-OR WORDS. XORs the source word operand with the destination word operand and stores the result in the destination operand. The result has ones in the bit positions in which either operand (but not both) had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) XOR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XOR wreg, waop
(100001aa) (waop) (wreg)</td></tr>
<tr><td>XORB</td><td>LOGICAL EXCLUSIVE-OR BYTES. XORs the source byte operand with the destination byte operand and stores the result in the destination operand. The result has ones in the bit positions in which either operand (but not both) had a “1” and zeros in all other bit positions.
(DEST) ← (DEST) XOR (SRC)
<table><tr><th colspan="6">PSW Flag Settings</th></tr><tr><th>Z</th><th>N</th><th>C</th><th>V</th><th>VT</th><th>ST</th></tr><tr><td>✓</td><td>✓</td><td>0</td><td>0</td><td>—</td><td>—</td></tr></table></td><td>DEST, SRC
XORB breg, baop
(100101aa) (baop) (breg)</td></tr>
</table>

Table A-7 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics.

Table A-7. Instruction Opcodes

Hex Code	Instruction Mnemonic
00	SKIP
01	CLR
02	NOT
03	NEG
04	XCH Direct
05	DEC
06	EXT
07	INC
08	SHR
09	SHL
0A	SHRA
0B	XCH Indexed
0C	SHRL
0D	SHLL
0E	SHRAL
0F	NORML
10	Reserved
11	CLRB
12	NOTB
13	NEGB
14	XCHB Direct
15	DECB
16	EXTB
17	INCB
18	SHRB
19	SHLB
1A	SHRAB
1B	XCHB Indexed
1C	EST Indirect
1D	EST Indexed
1E	ESTB Indirect
1F	ESTB Indexed
20–27	SJMP
28–2F	SCALL
30–37	JBC
38–3F	JBS
40	AND Direct (3 ops)
41	AND Immediate (3 ops)
42	AND Indirect (3 ops)
43	AND Indexed (3 ops)

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
44	ADD Direct (3 ops)
45	ADD Immediate (3 ops)
46	ADD Indirect (3 ops)
47	ADD Indexed (3 ops)
48	SUB Direct (3 ops)
49	SUB Immediate (3 ops)
4A	SUB Indirect (3 ops)
4B	SUB Indexed (3 ops)
4C	MULU Direct (3 ops)
4D	MULU Immediate (3 ops)
4E	MULU Indirect (3 ops)
4F	MULU Indexed (3 ops)
50	ANDB Direct (3 ops)
51	ANDB Immediate (3 ops)
52	ANDB Indirect (3 ops)
53	ANDB Indexed (3 ops)
54	ADDB Direct (3 ops)
55	ADDB Immediate (3 ops)
56	ADDB Indirect (3 ops)
57	ADDB Indexed (3 ops)
58	SUBB Direct (3 ops)
59	SUBB Immediate (3 ops)
5A	SUBB Indirect (3 ops)
5B	SUBB Indexed (3 ops)
5C	MULUB Direct (3 ops)
5D	MULUB Immediate (3 ops)
5E	MULUB Indirect (3 ops)
5F	MULUB Indexed (3 ops)
60	AND Direct (2 ops)
61	AND Immediate (2 ops)
62	AND Indirect (2 ops)
63	AND Indexed (2 ops)
64	ADD Direct (2 ops)
65	ADD Immediate (2 ops)
66	ADD Indirect (2 ops)
67	ADD Indexed (2 ops)
68	SUB Direct (2 ops)
69	SUB Immediate (2 ops)
6A	SUB Indirect (2 ops)
6B	SUB Indexed (2 ops)
6C	MULU Direct (2 ops)

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
6D	MULU Immediate (2 ops)
6E	MULU Indirect (2 ops)
6F	MULU Indexed (2 ops)
70	ANDB Direct (2 ops)
71	ANDB Immediate (2 ops)
72	ANDB Indirect (2 ops)
73	ANDB Indexed (2 ops)
74	ADDB Direct (2 ops)
75	ADDB Immediate (2 ops)
76	ADDB Indirect (2 ops)
77	ADDB Indexed (2 ops)
78	SUBB Direct (2 ops)
79	SUBB Immediate (2 ops)
7A	SUBB Indirect (2 ops)
7B	SUBB Indexed (2 ops)
7C	MULUB Direct (2 ops)
7D	MULUB Immediate (2 ops)
7E	MULUB Indirect (2 ops)
7F	MULUB Indexed (2 ops)
80	OR Direct
81	OR Immediate
82	OR Indirect
83	OR Indexed
84	XOR Direct
85	XOR Immediate
86	XOR Indirect
87	XOR Indexed
88	CMP Direct
89	CMP Immediate
8A	CMP Indirect
8B	CMP Indexed
8C	DIVU Direct
8E	DIVU Indirect
8F	DIVU Indexed
90	ORB Direct
91	ORB Immediate
92	ORB Indirect
93	ORB Indexed
94	XORB Direct
95	XORB Immediate
96	XORB Indirect

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
97	XORB Indexed
98	CMPB Direct
99	CMPB Immediate
9A	CMPB Indirect
9B	CMPB Indexed
9C	DIVUB Direct
9D	DIVUB Immediate
9E	DIVUB Indirect
9F	DIVUB Indexed
A0	LD Direct
A1	LD Immediate
A2	LD Indirect
A3	LD Indexed
A4	ADDC Direct
A5	ADDC Immediate
A6	ADDC Indirect
A7	ADDC Indexed
A8	SUBC Direct
A9	SUBC Immediate
AA	SUBC Indirect
AB	SUBC Indexed
AC	LDBZE Direct
AD	LDBZE Immediate
AE	LDBZE Indirect
AF	LDBZE Indexed
B0	LDB Direct
B1	LDB Immediate
B2	LDB Indirect
B3	LDB Indexed
B4	ADDCB Direct
B5	ADDCB Immediate
B6	ADDCB Indirect
B7	ADDCB Indexed
B8	SUBCB Direct
B9	SUBCB Immediate
BA	SUBCB Indirect
BB	SUBCB Indexed
BC	LDBSE Direct
BD	LDBSE Immediate
BE	LDBSE Indirect
BF	LDBSE Indexed

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
C0	ST Direct
C1	BMOV
C2	ST Indirect
C3	ST Indexed
C4	STB Direct
C5	CMPL
C6	STB Indirect
C7	STB Indexed
C8	PUSH Direct
C9	PUSH Immediate
CA	PUSH Indirect
CB	PUSH Indexed
CC	POP Direct
CD	BMOVI
CE	POP Indirect
CF	POP Indexed
D0	JNST
D1	JNH
D2	JGT
D3	JNC
D4	JNVT
D5	JNV
D4	JNVT
D5	JNV
D6	JGE
D7	JNE
D8	JST
D9	JH
DA	JLE
DB	JC
DC	JVT
DD	JV
DE	JLT
DF	JE
E0	DJNZ
E1	DJNZW
E2	TIJMP
E3	BR Indirect, 64-Kbyte mode
	EBR Indirect, 1-Mbyte mode
E4	EBMOVI
E5	Reserved

Table A-7. Instruction Opcodes (Continued)

Hex Code	Instruction Mnemonic
E6	EJMP
E7	LJMP
E8	ELD Indirect
E9	ELD Indexed
EA	ELDB Indirect
EB	ELDB Indexed
EC	DPTS
ED	EPTS
EE	Reserved (Note 1)
EF	LCALL
F0	RET
F1	ECALL
F2	PUSHF
F3	POPF
F4	PUSHA
F5	POPA
F6	IDLPD
F7	TRAP
F8	CLRC
F9	SETC
FA	DI
FB	EI
FC	CLRVT
FD	NOP
FE	DIV/DIVB/MUL/MULB (Note 2)
FF	RST

NOTES:
1. This opcode is reserved, but it does not generate an unimplemented opcode interrupt.
2. Signed multiplication and division are two-byte instructions. For each signed instruction, the first byte is "FE" and the second is the opcode of the corresponding unsigned instruction. For example, the opcode for MULU (3 operands) direct is "4C," so the opcode for MUL (3 operands) direct is "FE 4C."

Table A-8 lists instructions along with their lengths and opcodes for each applicable addressing mode. A dash (—) in any column indicates "not applicable."

Table A-8. Instruction Lengths and Hexadecimal Opcodes

Arithmetic (Group I)								
Mnemonic	Direct		Immediate		Indirect (Note 1)		Indexed (Notes 1, 2)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length S/L	Opcode
ADD (2 ops)	3	64	4	65	3	66	4/5	67
ADD (3 ops)	4	44	5	45	4	46	5/6	47
ADDB (2 ops)	3	74	3	75	3	76	4/5	77
ADDB (3 ops)	4	54	4	55	4	56	5/6	57
ADDC	3	A4	4	A5	3	A6	4/5	A7
ADDCB	3	B4	3	B5	3	B6	4/5	B7
CLR	2	01	—	—	—	—	—	—
CLRB	2	11	—	—	—	—	—	—
CMP	3	88	4	89	3	8A	4/5	8B
CMPB	3	98	3	99	3	9A	4/5	9B
CMPL	3	C5	—	—	—	—	—	—
DEC	2	05	—	—	—	—	—	—
DECB	2	15	—	—	—	—	—	—
EXT	2	06	—	—	—	—	—	—
EXTB	2	16	—	—	—	—	—	—
INC	2	07	—	—	—	—	—	—
INCB	2	17	—	—	—	—	—	—
SUB (2 ops)	3	68	4	69	3	6A	4/5	6B
SUB (3 ops)	4	48	5	49	4	4A	5/6	4B
SUBB (2 ops)	3	78	3	79	3	7A	4/5	7B
SUBB (3 ops)	4	58	4	59	4	5A	5/6	5B
SUBC	3	A8	4	A9	3	AA	4/5	AB
SUBCB	3	B8	3	B9	3	BA	4/5	BB

NOTES:

1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-8. Instruction Lengths and Hexadecimal Opcodes (Continued)

Arithmetic (Group II)								
Mnemonic	Direct		Immediate		Indirect (Note 1)		Indexed (Notes 1, 2)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length S/L	Opcode
DIV	4	FE 8C	5	FE 8D	4	FE 8E	5/6	FE 8F
DIVB	4	FE 9C	4	FE 9D	4	FE 9E	5/6	FE 9F
DIVU	3	8C	4	8D	3	8E	4/5	8F
DIVUB	3	9C	3	9D	3	9E	4/5	9F
MUL (2 ops)	4	FE 6C	5	FE 6D	4	FE 6E	5/6	FE 6F
MUL (3 ops)	5	FE 4C	6	FE 4D	5	FE 4E	6/7	FE 4F
MULB (2 ops)	4	FE 7C	4	FE 7D	4	FE 7E	5/6	FE 7F
MULB (3 ops)	5	FE 5C	5	FE 5D	5	FE 5E	6/7	FE 5F
MULU (2 ops)	3	6C	4	6D	3	6E	4/5	6F
MULU (3 ops)	4	4C	5	4D	4	4E	5/6	4F
MULUB (2 ops)	3	7C	3	7D	3	7E	4/5	7F
MULUB (3 ops)	4	5C	4	5D	4	5E	5/6	5F
Logical								
Mnemonic	Direct		Immediate		Indirect (Note 1)		Indexed (Notes 1, 2)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length S/L	Opcode
AND (2 ops)	3	60	4	61	3	62	4/5	63
AND (3 ops)	4	40	5	41	4	42	5/6	43
ANDB (2 ops)	3	70	3	71	3	72	4/5	73
ANDB (3 ops)	4	50	4	51	4	52	5/6	53
NEG	2	03	—	—	—	—	—	—
NEGB	2	13	—	—	—	—	—	—
NOT	2	02	—	—	—	—	—	—
NOTB	2	12	—	—	—	—	—	—
OR	3	80	4	81	3	82	4/5	83
ORB	3	90	3	91	3	92	4/5	93
XOR	3	84	4	85	3	86	4/5	87
XORB	3	94	3	95	3	96	4/5	97

NOTES:

1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-8. Instruction Lengths and Hexadecimal Opcodes (Continued)

Stack								
Mnemonic	Direct		Immediate		Indirect (Note 1)		Indexed (Notes 1, 2)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length S/L	Opcode
POP	2	CC	—	—	2	CE	3/4	CF
POPA	1	F5	—	—	—	—	—	—
POPF	1	F3	—	—	—	—	—	—
PUSH	2	C8	3	C9	2	CA	3/4	CB
PUSHA	1	F4	—	—	—	—	—	—
PUSHF	1	F2	—	—	—	—	—	—

NOTES:
1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-8. Instruction Lengths and Hexadecimal Opcodes (Continued)

Data								
Mnemonic	**Direct**		**Immediate**		**Extended-indirect**		**Extended-indexed**	
	Length	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**
EBMOVI	—	—	—	—	3	E4	—	—
ELD	—	—	—	—	3	E8	6	E9
ELDB	—	—	—	—	3	EA	6	EB
EST	—	—	—	—	3	1C	6	1D
ESTB	—	—	—	—	3	1E	6	1F
Mnemonic	**Direct**		**Immediate**		**Indirect (Note 1)**		**Indexed (Notes 1, 2)**	
	Length	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**	**Length S/L**	**Opcode**
BMOV	—	—	—	—	3	C1	—	—
BMOVI	—	—	—	—	3	CD	—	—
LD	3	A0	4	A1	3	A2	4/5	A3
LDB	3	B0	3	B1	3	B2	4/5	B3
LDBSE	3	BC	3	BD	3	BE	4/5	BF
LDBZE	3	AC	3	AD	3	AE	4/5	AF
ST	3	C0	—	—	3	C2	4/5	C3
STB	3	C4	—	—	3	C6	4/5	C7
XCH	3	04	—	—	—	—	4/5	0B
XCHB	3	14	—	—	—	—	4/5	1B

NOTES:
1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-8. Instruction Lengths and Hexadecimal Opcodes (Continued)

Jump								
Mnemonic	**Direct**		**Immediate**		**Extended-indirect**		**Extended-indexed**	
	Length	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**
EBR	—	—	—	—	2	E3	—	—
EJMP	—	—	—	—	—	—	4	E6
Mnemonic	**Direct**		**Immediate**		**Indirect (Note 1)**		**Indexed (Notes 1, 2)**	
	Length	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**	**Length S/L**	**Opcode**
BR	—	—	—	—	2	E3	—	—
LJMP	—	—	—	—	—	—	—/3	E7
SJMP (Note 3)	—	—	—	—	—	—	2/—	20–27
TIJMP	4	E2	4	E2	—	—	—/4	E2
Call								
Mnemonic	**Direct**		**Immediate**		**Extended-indirect**		**Extended-indexed**	
	Length	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**
ECALL	—	—	—	—	—	—	4	F1
Mnemonic	**Direct**		**Immediate**		**Indirect (Note 1)**		**Indexed (Note 1)**	
	Length	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**	**Length**	**Opcode**
LCALL	—	—	—	—	—	—	3	EF
RET	—	—	—	—	1	F0	—	—
SCALL (Note 3)	—	—	—	—	—	—	2	28–2F
TRAP	1	F7	—	—	—	—	—	—

NOTES:

1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-8. Instruction Lengths and Hexadecimal Opcodes (Continued)

Conditional Jump								
Mnemonic	Direct		Immediate		Indirect		Indexed (Notes 1, 2)	
	Length	Opcode	Length	Opcode	Length	Opcode	Length S/L	Opcode
DJNZ	—	—	—	—	—	—	3/—	E0
DJNZW	—	—	—	—	—	—	3/—	E1
JBC	—	—	—	—	—	—	3/—	30–37
JBS	—	—	—	—	—	—	3/—	38–3F
JC	—	—	—	—	—	—	2/—	DB
JE	—	—	—	—	—	—	2/—	DF
JGE	—	—	—	—	—	—	2/—	D6
JGT	—	—	—	—	—	—	2/—	D2
JH	—	—	—	—	—	—	2/—	D9
JLE	—	—	—	—	—	—	2/—	DA
JLT	—	—	—	—	—	—	2/—	DE
JNC	—	—	—	—	—	—	2/—	D3
JNE	—	—	—	—	—	—	2/—	D7
JNH	—	—	—	—	—	—	2/—	D1
JNST	—	—	—	—	—	—	2/—	D0
JNV	—	—	—	—	—	—	2/—	D5
JNVT	—	—	—	—	—	—	2/—	D4
JST	—	—	—	—	—	—	2/—	D8
JV	—	—	—	—	—	—	2/—	DD
JVT	—	—	—	—	—	—	2/—	DC

NOTES:
1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-8. Instruction Lengths and Hexadecimal Opcodes (Continued)

Shift								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	Opcode
NORML	3	0F	—	—	—	—	—	—
SHL	3	09	—	—	—	—	—	—
SHLB	3	19	—	—	—	—	—	—
SHLL	3	0D	—	—	—	—	—	—
SHR	3	08	—	—	—	—	—	—
SHRA	3	0A	—	—	—	—	—	—
SHRAB	3	1A	—	—	—	—	—	—
SHRAL	3	0E	—	—	—	—	—	—
SHRB	3	18	—	—	—	—	—	—
SHRL	3	0C	—	—	—	—	—	—
Special								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	Opcode
CLRC	1	F8	—	—	—	—	—	—
CLRVT	1	FC	—	—	—	—	—	—
DI	1	FA	—	—	—	—	—	—
EI	1	FB	—	—	—	—	—	—
IDLPD	—	—	1	F6	—	—	—	—
NOP	1	FD	—	—	—	—	—	—
RST	1	FF	—	—	—	—	—	—
SETC	1	F9	—	—	—	—	—	—
SKIP	2	00	—	—	—	—	—	—
PTS								
Mnemonic	Direct		Immediate		Indirect		Indexed	
	Length	Opcode	Length	Opcode	Length	Opcode	Length	Opcode
DPTS	1	EC	—	—	—	—	—	—
EPTS	1	ED	—	—	—	—	—	—

NOTES:
1. Indirect normal and indirect autoincrement share the same opcodes, as do short- and long-indexed modes. Because word registers always have even addresses, the address can be expressed in the upper seven bits; the least-significant bit determines the addressing mode. Indirect normal and short-indexed modes make the second byte of the instruction even (LSB = 0). Indirect autoincrement and long-indexed modes make the second byte odd (LSB = 1).
2. For indexed instructions, the first column lists instruction lengths as *S*/*L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length.
3. For the SCALL and SJMP instructions, the three least-significant bits of the opcode are concatenated with the eight bits to form an 11-bit, 2's complement offset.

Table A-9 lists instructions alphabetically within groups, along with their execution times, expressed in state times.

Table A-9. Instruction Execution Times (in State Times)

Arithmetic (Group I)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
ADD (2 ops)	4	5	6	8	7	9	6	8	7	9
ADD (3 ops)	5	6	7	10	8	11	7	10	8	11
ADDB (2 ops)	4	4	6	8	7	9	6	8	7	9
ADDB (3 ops)	5	5	7	10	8	11	7	10	8	11
ADDC	4	5	6	8	7	9	6	8	7	9
ADDCB	4	4	6	8	7	9	6	8	7	9
CLR	3	—	—	—	—	—	—	—	—	—
CLRB	3	—	—	—	—	—	—	—	—	—
CMP	4	5	6	8	7	9	6	8	7	9
CMPB	4	4	6	8	7	9	6	8	7	9
CMPL	7	—	—	—	—	—	—	—	—	—
DEC	3	—	—	—	—	—	—	—	—	—
DECB	3	—	—	—	—	—	—	—	—	—
EXT	4	—	—	—	—	—	—	—	—	—
EXTB	4	—	—	—	—	—	—	—	—	—
INC	3	—	—	—	—	—	—	—	—	—
INCB	3	—	—	—	—	—	—	—	—	—
SUB (2 ops)	4	5	6	8	7	9	6	8	7	9
SUB (3 ops)	5	6	7	10	8	11	7	10	8	11
SUBB (2 ops)	4	4	6	8	7	9	6	8	7	9
SUBB (3 ops)	5	5	7	10	8	11	7	10	8	11
SUBC	4	5	6	8	7	9	6	8	7	9
SUBCB	4	4	6	8	7	9	6	8	7	9

NOTE: The column entitled "Reg." lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled "Mem." lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Arithmetic (Group II)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
DIV	26	27	28	31	29	32	29	32	30	33
DIVB	18	18	20	23	21	24	21	24	22	25
DIVU	24	25	26	29	27	30	27	30	28	31
DIVUB	16	16	18	21	19	22	19	22	20	23
MUL (2 ops)	16	17	18	21	19	22	19	22	20	23
MUL (3 ops)	16	17	18	21	19	22	19	22	20	23
MULB (2 ops)	12	12	14	17	15	18	15	18	16	19
MULB (3 ops)	12	12	14	17	15	18	15	18	16	19
MULU (2 ops)	14	15	16	19	17	19	17	20	18	21
MULU (3 ops)	14	15	16	19	17	19	17	20	18	21
MULUB (2 ops)	10	10	12	15	13	15	12	16	14	17
MULUB (3 ops)	10	10	12	15	13	15	12	16	14	17
Logical										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
AND (2 ops)	4	5	6	8	7	9	6	8	7	9
AND (3 ops)	5	6	7	10	8	11	7	10	8	11
ANDB (2 ops)	4	4	6	8	7	9	6	8	7	9
ANDB (3 ops)	5	5	7	10	8	11	7	10	8	11
NEG	3	—	—	—	—	—	—	—	—	—
NEGB	3	—	—	—	—	—	—	—	—	—
NOT	3	—	—	—	—	—	—	—	—	—
NOTB	3	—	—	—	—	—	—	—	—	—
OR	4	5	6	8	7	9	6	8	7	9
ORB	4	4	6	8	7	9	6	8	7	9
XOR	4	5	6	8	7	9	6	8	7	9
XORB	4	4	6	8	7	9	6	8	7	9

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Stack (Register)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
POP	8	—	10	12	11	13	11	13	12	14
POPA	12	—	—	—	—	—	—	—	—	—
POPF	7	—	—	—	—	—	—	—	—	—
PUSH	6	7	9	12	10	13	10	13	11	14
PUSHA	12	—	—	—	—	—	—	—	—	—
PUSHF	6	—	—	—	—	—	—	—	—	—
Stack (Memory)										
Mnemonic	Direct	Immed.	Indirect				Indexed			
			Normal		Autoinc.		Short		Long	
			Reg.	Mem.	Reg.	Mem.	Reg.	Mem.	Reg.	Mem.
POP	11	—	13	15	14	16	14	16	15	17
POPA	18	—	—	—	—	—	—	—	—	—
POPF	10	—	—	—	—	—	—	—	—	—
PUSH	8	9	11	14	12	15	12	15	13	16
PUSHA	18	—	—	—	—	—	—	—	—	—
PUSHF	8	—	—	—	—	—	—	—	—	—

NOTE: The column entitled "Reg." lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled "Mem." lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Data										
Mnemonic	**Extended-indirect (Normal)**									
EBMOVI	register/register 8 + 14 per word + 16 per interrupt memory/register 8 + 17 per word + 16 per interrupt memory/memory 8 + 20 per word + 16 per interrupt									
Mnemonic	**Indirect**									
BMOV	register/register 6 + 8 per word memory/register 6 + 11 per word memory/memory 6 + 14 per word									
BMOVI	register/register 7 + 8 per word + 14 per interrupt memory/register 7 + 11 per word + 14 per interrupt memory/memory 7 + 14 per word + 14 per interrupt									
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**				**Extended-indexed**			
			Normal		**Autoinc.**					
ELD	—	—	6	9	8	11	8		11	
ELDB	—	—	6	9	8	11	8		11	
EST	—	—	6	9	8	11	8		11	
ESTB	—	—	6	9	8	11	8		11	
Mnemonic	**Direct**	**Immed.**	**Indirect**				**Indexed**			
			Normal		**Autoinc.**		**Short**		**Long**	
			Reg.	**Mem.**	**Reg.**	**Mem.**	**Reg.**	**Mem.**	**Reg.**	**Mem.**
LD	4	5	5	8	6	8	6	9	7	10
LDB	4	4	5	8	6	8	6	9	7	10
LDBSE	4	4	5	8	6	8	6	9	7	10
LDBZE	4	4	5	8	6	8	6	9	7	10
ST	4	—	5	8	6	9	6	9	7	10
STB	4	—	5	8	6	8	6	9	7	10
XCH	5	—	—	—	—	—	8	13	9	14
XCHB	5	—	—	—	—	—	8	13	9	14

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Jump						
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**		**Extended-indexed**	
			Normal	**Autoinc.**		
EBR	—	—	9	—	—	
EJMP	—	—	—	—	8	
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
BR	—	—	7	7	—	—
LJMP	—	—	—	—	—	7
SJMP	—	—	—	—	7	—
TIJMP register/register memory/register memory/memory	—	—	 15 18 21	—	—	—
Call (Register)						
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**		**Extended-indexed**	
			Normal	**Autoinc.**		
ECALL 1-Mbyte mode	—	—	—	—	16	
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
LCALL 1-Mbyte mode 64-Kbyte mode	—	—	—	—	—	 15 11
RET 1-Mbyte mode 64-Kbyte mode	—	—	 16 11	—	—	—
SCALL 1-Mbyte mode 64-Kbyte mode	—	—	—	—	—	 15 11
TRAP 1-Mbyte mode 64-Kbyte mode	 19 16	—	—	—	—	—

NOTE: The column entitled "Reg." lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled "Mem." lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Call (Memory)						
Mnemonic	**Direct**	**Immed.**	**Extended-indirect**		**Extended-indexed**	
			Normal	**Autoinc.**		
ECALL 1-Mbyte mode	—	—	—	—	22	
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
LCALL 1-Mbyte mode 64-Kbyte mode	—	—	—	—	—	18 13
RET 1-Mbyte mode 64-Kbyte mode	—	—	22 14	—	—	—
SCALL 1-Mbyte mode 64-Kbyte mode	—	—	—	—	—	18 13
TRAP 1-Mbyte mode 64-Kbyte mode	25 18	—	—	—	—	—

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Conditional Jump	
Mnemonic	**Short-Indexed**
DJNZ	5 (jump not taken), 9 (jump taken)
DJNZW	6 (jump not taken), 10 (jump taken)
JBC	5 (jump not taken), 9 (jump taken)
JBS	5 (jump not taken), 9 (jump taken)
JC	4 (jump not taken), 8 (jump taken)
JE	4 (jump not taken), 8 (jump taken)
JGE	4 (jump not taken), 8 (jump taken)
JGT	4 (jump not taken), 8 (jump taken)
JH	4 (jump not taken), 8 (jump taken)
JLE	4 (jump not taken), 8 (jump taken)
JLT	4 (jump not taken), 8 (jump taken)
JNC	4 (jump not taken), 8 (jump taken)
JNE	4 (jump not taken), 8 (jump taken)
JNH	4 (jump not taken), 8 (jump taken)
JNST	4 (jump not taken), 8 (jump taken)
JNV	4 (jump not taken), 8 (jump taken)
JNVT	4 (jump not taken), 8 (jump taken)
JST	4 (jump not taken), 8 (jump taken)
JV	4 (jump not taken), 8 (jump taken)
JVT	4 (jump not taken), 8 (jump taken)
Shift	
Mnemonic	**Direct**
NORML	8 + 1 per shift (9 for 0 shift)
SHL	6 + 1 per shift (7 for 0 shift)
SHLB	6 + 1 per shift (7 for 0 shift)
SHLL	7 + 1 per shift (8 for 0 shift)
SHR	6 + 1 per shift (7 for 0 shift)
SHRA	6 + 1 per shift (7 for 0 shift)
SHRAB	6 + 1 per shift (7 for 0 shift)
SHRAL	7 + 1 per shift (8 for 0 shift)
SHRB	6 + 1 per shift (7 for 0 shift)
SHRL	7 + 1 per shift (8 for 0 shift)

NOTE: The column entitled "Reg." lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled "Mem." lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

Table A-9. Instruction Execution Times (in State Times) (Continued)

Special						
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
CLRC	2	—	—	—	—	—
CLRVT	2	—	—	—	—	—
DI	2	—	—	—	—	—
EI	2	—	—	—	—	—
IDLPD Valid key Invalid key	 — —	 12 28	 — —	 — —	 — —	 — —
NOP	2	—	—	—	—	—
RST	4	—	—	—	—	—
SETC	2	—	—	—	—	—
SKIP	3	—	—	—	—	—
PTS						
Mnemonic	**Direct**	**Immed.**	**Indirect**		**Indexed**	
			Normal	**Autoinc.**	**Short**	**Long**
DPTS	2	—	—	—	—	—
EPTS	2	—	—	—	—	—

NOTE: The column entitled “Reg.” lists the instruction execution times for accesses to the register file or peripheral SFRs. The column entitled “Mem.” lists the instruction execution times for accesses to all memory-mapped registers, I/O, or memory. See Table 5-1 on page 5-4 for address information.

B

Signal Descriptions

APPENDIX B
SIGNAL DESCRIPTIONS

This appendix provides reference information for the pin functions of the 8XC196NP and 80C196NU.

B.1 FUNCTIONAL GROUPINGS OF SIGNALS

Table B-1 lists the signals for the 8XC196NP and 80C196NU, grouped by function. A diagram of each package that is currently available shows the pin location of each signal.

NOTE

As new packages are supported, they will be added to the datasheets first. If your package type is not shown in this appendix, refer to the latest datasheet to find the pin locations.

Table B-1. 8XC196NP and 80C196NU Signals Arranged by Function

Address & Data
A19:0
AD15:0

Power & Ground
V_{CC}
V_{SS}

Processor Control
EA# (NP only)
EXTINT3:0
NMI
ONCE
PLLEN1 (NU only)
PLLEN2 (NU only)
RESET#
RPD
XTAL1
XTAL2

Input/Output
EPORT3:0
P1.3:0/EPA3:0
P1.4/T1CLK
P1.5/T1DIR
P1.6/T2CLK
P1.7/T2DIR
P2.0/TXD
P2.1/RXD
P2.7:2
P3.7:0
P4.2:0/PWM2:0
P4.3

Bus Control & Status
ALE
BHE#/WRH#
BREQ#
CLKOUT
CS5:0#
HOLD#
HLDA#
INST
RD#
READY
WR#/WRL#

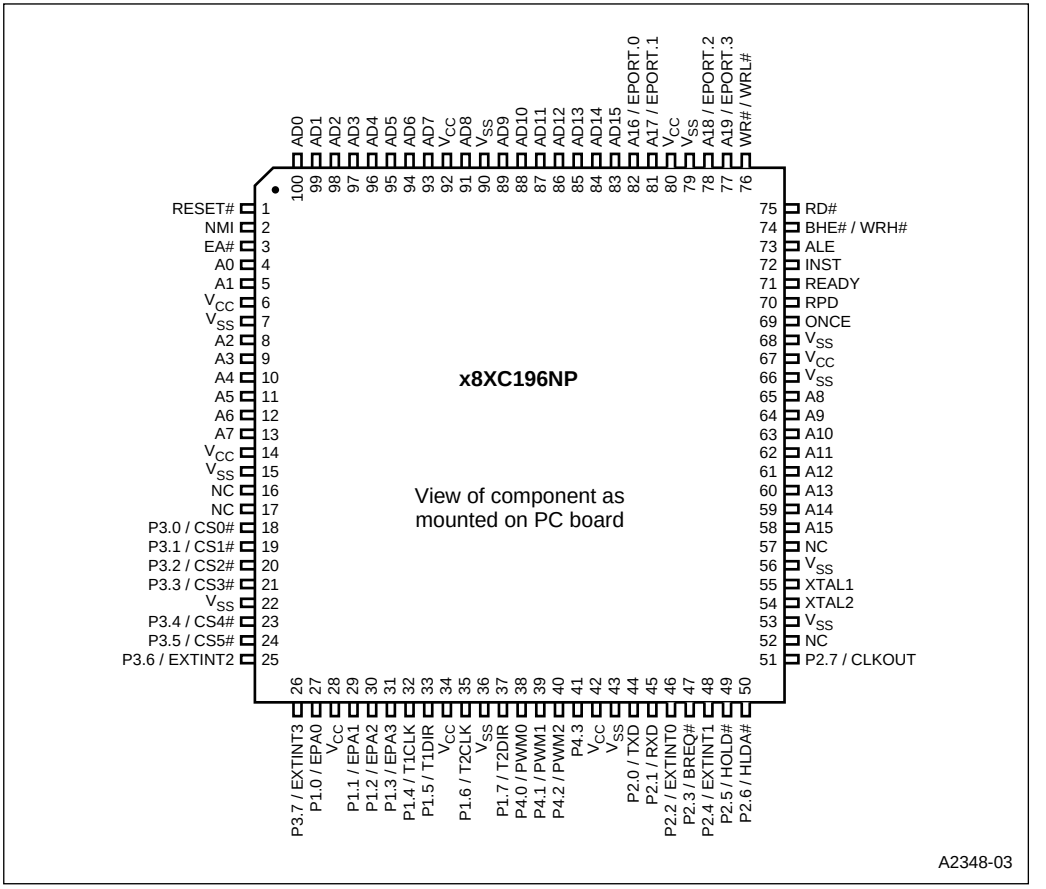


Figure B-1. 8XC196NP 100-lead SQFP Package

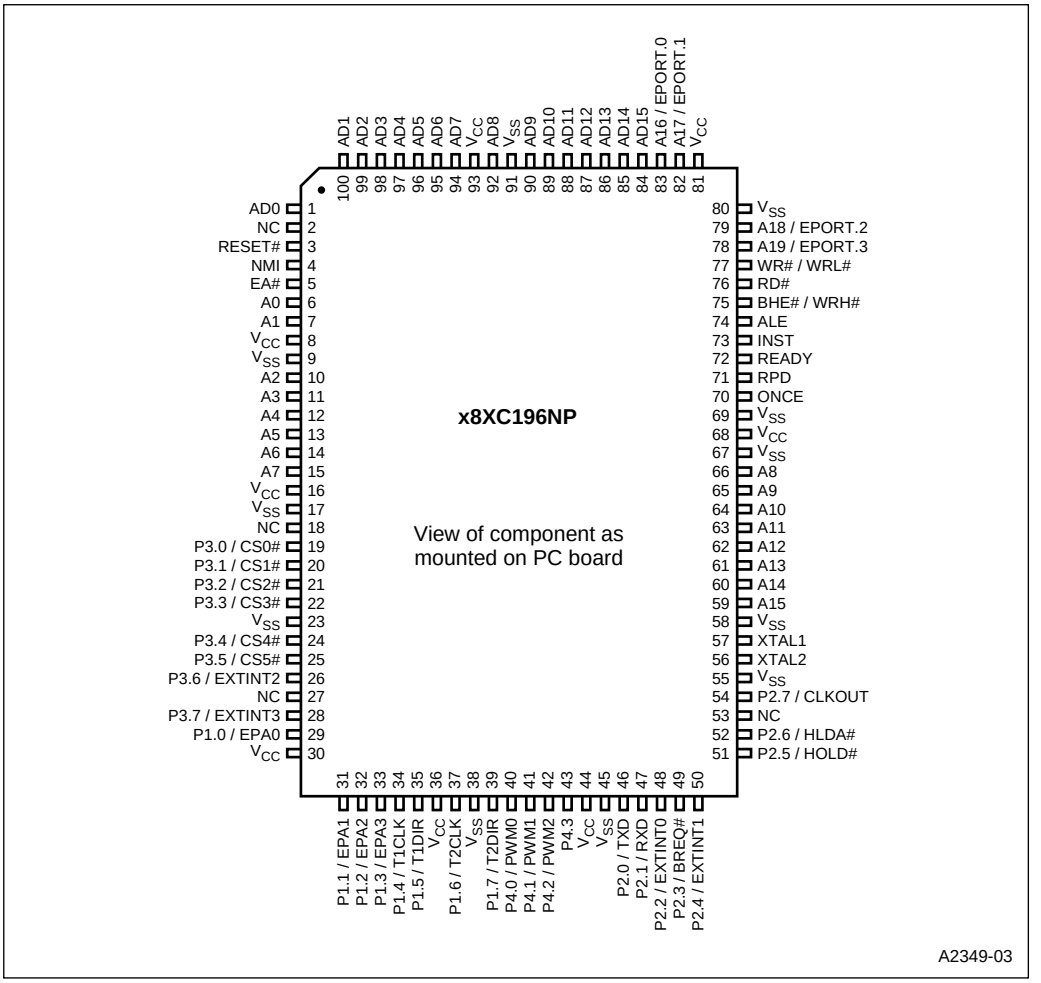


Figure B-2. 8XC196NP 100-lead QFP Package

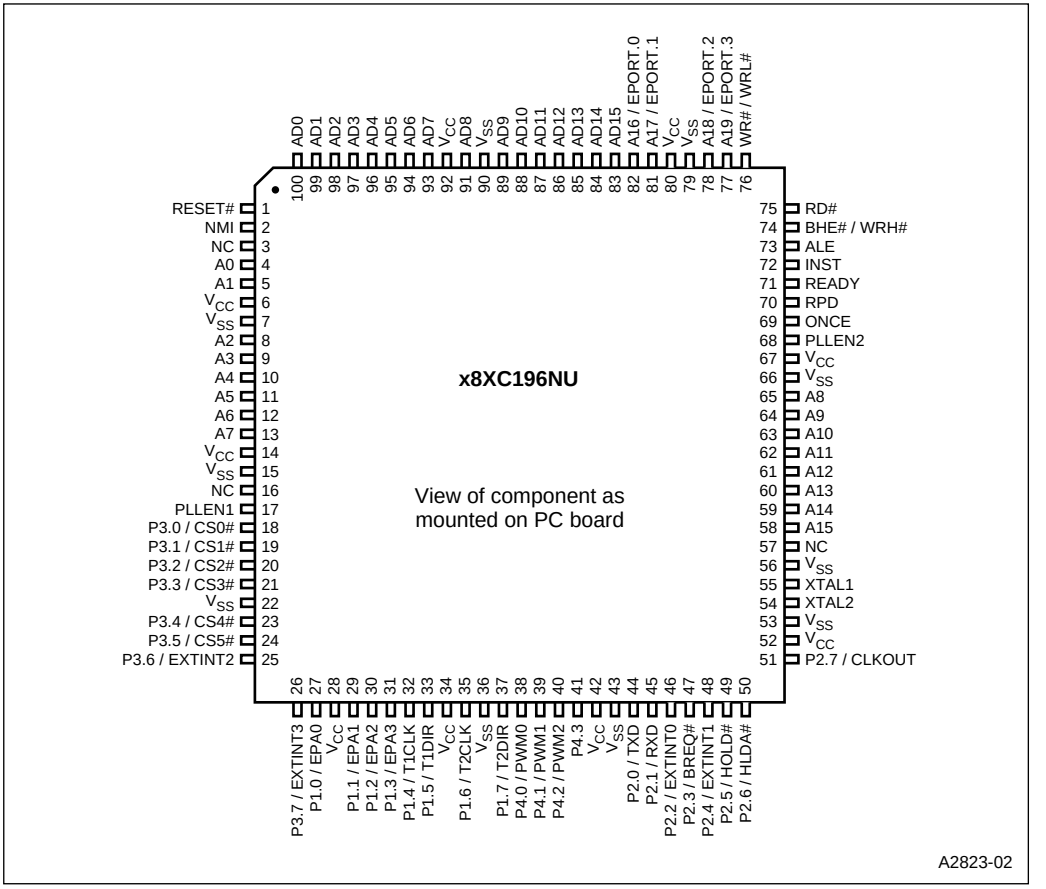


Figure B-3. 80C196NU 100-lead SQFP Package

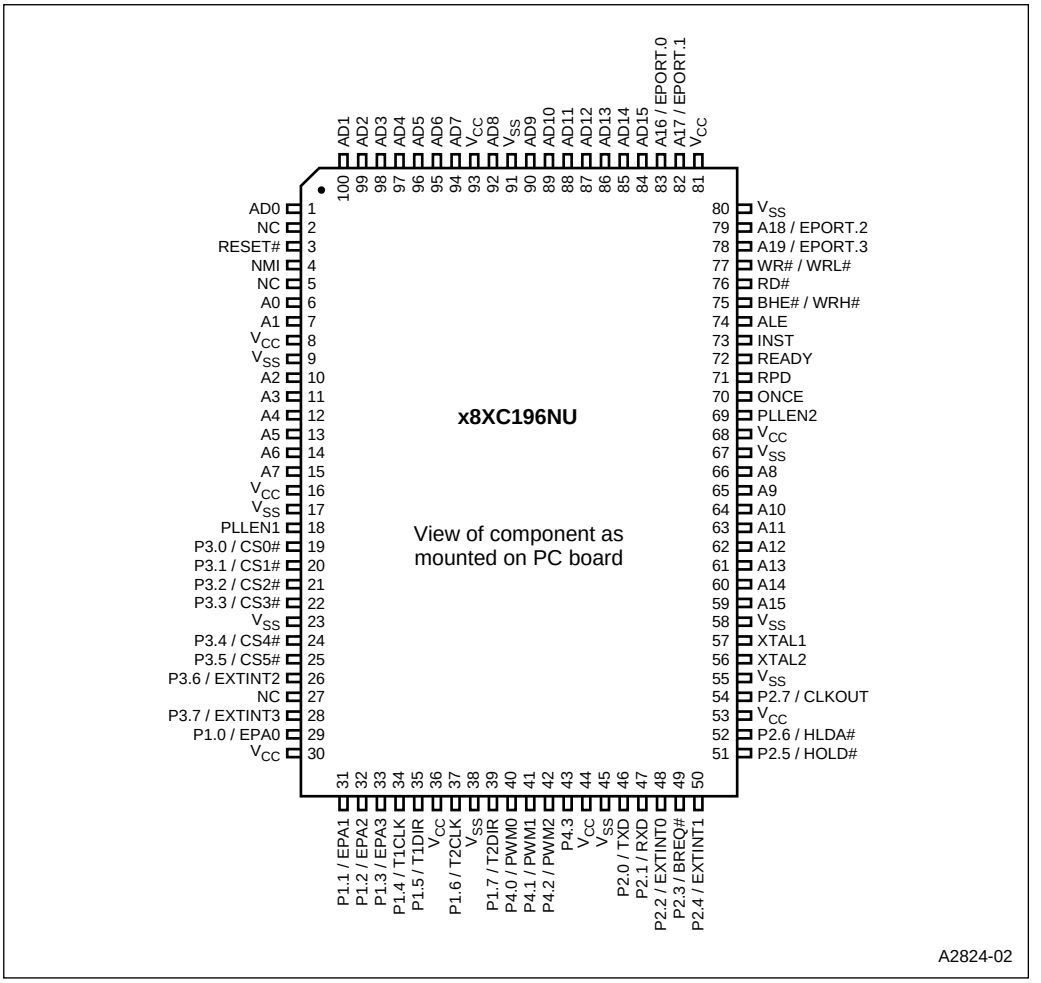


Figure B-4. 80C196NU 100-lead QFP Package

B.2 SIGNAL DESCRIPTIONS

Table B-2 defines the columns used in Table B-3, which describes the signals.

Table B-2. Description of Columns of Table B-3

Column Heading	Description
Name	Lists the signals, arranged alphabetically. Many pins have two functions, so there are more entries in this column than there are pins. Every signal is listed in this column.
Type	Identifies the pin function listed in the *Name* column as an input (I), output (O), bidirectional (I/O), power (PWR), or ground (GND). Note that all inputs except RESET# are *sampled inputs*. RESET# is a level-sensitive input. During powerdown mode, the powerdown circuitry uses EXTINT*x* as a level-sensitive input.
Description	Briefly describes the function of the pin for the specific signal listed in the *Name* column. Also lists the alternate fuction that are multiplexed with the signal (if applicable).

Table B-3. Signal Descriptions

Name	Type	Description
A15:0	I/O	System Address Bus These address lines provide address bits 0–15 during the entire external memory cycle during both multiplexed and demultiplexed bus modes.
A19:16	I/O	Address Lines 16–19 These address lines provide address bits 16–19 during the entire external memory cycle, supporting extended addressing of the 1 Mbyte address space. **NOTE:** Internally, there are 24 address bits; however, only 20 address lines (A19:0) are bonded out. The internal address space is 16 Mbytes (000000–FFFFFFH) and the external address space is 1 Mbyte (00000–FFFFFH). The device resets to FF2080H in internal ROM or F2080H in external memory. A19:16 are multiplexed with EPORT.3:0.
AD15:0	I/O	Address/Data Lines The function of these pins depend on the bus size and mode. When a bus access is not occurring, these pins revert to their I/O port function. **16-bit Multiplexed Bus Mode**: AD15:0 drive address bits 0–15 during the first half of the bus cycle and drives or receives data during the second half of the bus cycle. **8-bit Multiplexed Bus Mode**: AD15:8 drive address bits 8–15 during the entire bus cycle. AD7:0 drive address bits 0–7 during the first half of the bus cycle and either drive or receive data during the second half of the bus cycle. **16-bit Demultiplexed Mode**: AD15:0 drive or receive data during the entire bus cycle. **8-bit Demultiplexed Mode**: AD7:0 drive or receive data during the entire bus cycle. AD15:8 drive the data that is currently on the high byte of the internal bus.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
ALE	O	Address Latch Enable This active-high output signal is asserted only during external memory cycles. ALE signals the start of an external bus cycle and indicates that valid address information is available on the system address/data bus (A19:16 and AD15:0 for a multiplexed bus; A19:0 for a demultiplexed bus). ALE differs from ADV# in that it does not remain active during the entire bus cycle. An external latch can use this signal to demultiplex address bits 0–15 from the address/data bus in multiplexed mode.
BHE#	O	Byte High Enable† During 16-bit bus cycles, this active-low output signal is asserted for word reads and writes and high-byte reads and writes to external memory. BHE# indicates that valid data is being transferred over the upper half of the system data bus. Use BHE#, in conjunction with A0, to determine which memory byte is being transferred over the system bus: **BHE# A0 Byte(s) Accessed** 0 0 both bytes 0 1 high byte only 1 0 low byte only BHE# is multiplexed with WRH#. † The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.
BREQ#	O	Bus Request This active-low output signal is asserted during a hold cycle when the bus controller has a pending external memory cycle. The device can assert BREQ# at the same time as or after it asserts HLDA#. Once it is asserted, BREQ# remains asserted until HOLD# is removed. You must enable the bus-hold protocol before using this signal (see “Enabling the Bus-hold Protocol” on page 13-32). BREQ# is multiplexed with P2.3.
CLKOUT	O	Clock Output Output of the internal clock generator. The CLKOUT frequency is ½ the internal operating frequency (f). CLKOUT has a 50% duty cycle. CLKOUT is multiplexed with P2.7.
CS5:0#	O	Chip-select Lines 0–5 The active-low output CS*x*# is asserted during an external memory cycle when the address to be accessed is in the range programmed for chip select *x*. If the external memory address is outside the range assigned to the six chip selects, no chip-select output is asserted and the bus configuration defaults to the CS5# values. Immediately following reset, CS0# is automatically assigned to the range FF2000–FF20FFH (F2000–F20FFH if external). CS5:0# is multiplexed with P3.5:0

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
EA# (NP only)	I	External Access This input determines whether memory accesses to special-purpose and program memory partitions (FF2000–FF2FFFH) are directed to internal or external memory. These accesses are directed to internal memory if EA# is held high and to external memory if EA# is held low. For an access to any other memory location, the value of EA# is irrelevant. EA# is not latched and can be switched dynamically during normal operating mode. Be sure to thoroughly consider the issues, such as different access times for internal and external memory, before using this dynamic switching capability. On devices with no internal nonvolatile memory, always connect EA# to V_{SS}. EA# is not implemented on the 80C196NU.
EPA3:0	I/O	Event Processor Array (EPA) Input/Output pins These are the high-speed input/output pins for the EPA capture/compare channels. For high-speed PWM applications, the outputs of two EPA channels (either EPA0 and EPA1 or EPA2 and EPA3) can be remapped to produce a PWM waveform on a shared output pin (see "Generating a High-speed PWM Output" on page 10-14). EPA3:0 are multiplexed with P1.3:0.
EPORT.3:0	I/O	Extended Addressing Port On the 8XC196NP, this is a 4-bit, bidirectional, memory-mapped I/O port. On the 8XC196NU, this is a 4-bit, bidirectional, standard I/O port. EPORT.3:0 are multiplexed with A19:16.
EXTINT3:0	I	External Interrupts In normal operating mode, a rising edge on EXTINT*x* sets the EXTINT*x* interrupt pending bit. EXTINT*x* is sampled during phase 2 (CLKOUT high). The minimum high time is one state time. In standby and powerdown modes, asserting the EXTINT*x* signal for at least 50 ns causes the device to resume normal operation. The interrupt need not be enabled, but the pin must be configured as a special-function input (see "Bidirectional Port Pin Configurations" on page 7-7). If the EXTINT*x* interrupt is enabled, the CPU executes the interrupt service routine. Otherwise, the CPU executes the instruction that immediately follows the command that invoked the power-saving mode. In idle mode, asserting any enabled interrupt causes the device to resume normal operation. EXTINT0 is multiplexed with P2.2, EXTINT1 is multiplexed with P2.4, EXTINT2 is multiplexed with P3.6, and EXTINT3 is multiplexed with P3.7.
HLDA#	O	Bus Hold Acknowledge This active-low output indicates that the CPU has released the bus as the result of an external device asserting HOLD#. HLDA# is multiplexed with P2.6.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
HOLD#	I	Bus Hold Request An external device uses this active-low input signal to request control of the bus. This pin functions as HOLD# only if the pin is configured for its special function (see “Bidirectional Port Pin Configurations” on page 7-7) and the bus-hold protocol is enabled. Setting bit 7 of the window selection register (WSR) enables the bus-hold protocol. HOLD# is multiplexed with P2.5.
INST	O	Instruction Fetch This active-high output signal is valid only during external memory bus cycles. When high, INST indicates that an instruction is being fetched from external memory. The signal remains high during the entire bus cycle of an external instruction fetch. INST is low for data accesses, including interrupt vector fetches and chip configuration byte reads. INST is low during internal memory fetches.
NMI	I	Nonmaskable Interrupt In normal operating mode, a rising edge on NMI generates a nonmaskable interrupt. NMI has the highest priority of all prioritized interrupts. Assert NMI for greater than one state time to guarantee that it is recognized.
ONCE	I	On-circuit Emulation Holding ONCE high during the rising edge of RESET# places the device into on-circuit emulation (ONCE) mode. This mode puts all pins into a high-impedance state, thereby isolating the device from other components in the system. The value of ONCE is latched when the RESET# pin goes inactive. While the device is in ONCE mode, you can debug the system using a clip-on emulator. To exit ONCE mode, reset the device by pulling the RESET# signal low. To prevent accidental entry into ONCE mode, connect the ONCE pin to V_{SS}.
P1.7:0	I/O	Port 1 This is a standard, bidirectional port that is multiplexed with individually selectable special-function signals. Port 1 is multiplexed as follows: P1.0/EPA0, P1.1/EPA1, P1.2/EPA2, P1.3/EPA3, P1.4/T1CLK, P1.5/T1DIR, P1.6/T2CLK, and P1.7/T2DIR.
P2.7:0	I/O	Port 2 This is a standard bidirectional port that is multiplexed with individually selectable special-function signals. Port 2 is multiplexed as follows: P2.0/TXD, P2.1/RXD, P2.2/EXTINT0, P2.3/BREQ#, P2.4/EXTINT1, P2.5/HOLD#, P2.6/HLDA#, and P2.7/CLKOUT.
P3.7:0	I/O	Port 3 This is an 8-bit, bidirectional, standard I/O port. Port 3 is multiplexed as follows: P3.0/CS0#, P3.1/CS1#, P3.2/CS2#, P3.3/CS3#, P3.4/CS4#, P3.5/CS5#, P3.6/EXTINT2, and P3.7/EXTINT3.
P4.3:0	I/O	Port 4 This is a 4-bit, bidirectional, standard I/O port with high-current drive capability. Port 4 is multiplexed as follows: P4.0/PWM0, P4.1/PWM1, and P4.2/PWM2. P4.3 is not multiplexed.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
PLLEN2:1 (NU only)	I	Phase-locked Loop 1 and 2 Enable These input pins are used to enable the on-chip clock multiplier feature and select either the doubled or quadrupled clock speed as follows: **PLLEN1** **PLLEN2** **Mode** 0 0 standard mode; clock multiplier circuitry disabled. Internal clock equals the XTAL1 input frequency. 0 1 Reserved† 1 0 doubled mode; clock multiplier circuitry enabled. Internal clock is twice the XTAL1 input frequency. 1 1 quadrupled mode; clock multiplier circuitry enabled. Internal clock is four times the XTAL1 input frequency. † This reserved combination causes the device to enter an unsupported test mode.
PWM2:0	O	Pulse Width Modulator Outputs These are PWM output pins with high-current drive capability. The duty cycle and frequency-pulse-widths are programmable. PWM2:0 are multiplexed with P4.2:0.
RD#	O	Read Read-signal output to external memory. RD# is asserted only during external memory reads.
READY	I	Ready Input This active-high input signal is used to lengthen external memory cycles for slow memory by generating wait states in addition to the wait states that are generated internally. When READY is high, CPU operation continues in a normal manner with wait states inserted as programmed in CCR0 or the chip-select *x* bus control register. READY is ignored for all internal memory accesses.
RESET#	I/O	Reset A level-sensitive reset input to and open-drain system reset output from the microcontroller. Either a falling edge on RESET# or an internal reset turns on a pull-down transistor connected to the RESET# pin for 16 state times. In the powerdown, standby, and idle modes, asserting RESET# causes the chip to reset and return to normal operating mode. After a device reset, the first instruction fetch is from FF2080H (or F2080H in external memory). For the 80C196NP and 80C196NU, the program and special-purpose memory locations (FF2000–FF2FFFH) reside in external memory. For the 83C196NP, these locations can reside either in external memory or in internal ROM.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
RPD	I	Return from Powerdown Timing pin for the return-from-powerdown circuit. If your application uses powerdown mode, connect a capacitor† between RPD and V_{SS} if **either** of the following conditions are true. • the internal oscillator is the clock source • the phase-locked loop (PLL) circuitry (80C196NU only) is enabled (see PLLEN2:1 signal description) The capacitor causes a delay that enables the oscillator and PLL circuitry to stabilize before the internal CPU and peripheral clocks are enabled. The capacitor is not required if your application uses powerdown mode and if **both** of the following conditions are true. • an external clock input is the clock source • the phase-locked loop circuitry (80C196NU only) is disabled If your application does not use powerdown mode, leave this pin unconnected. † Calculate the value of the capacitor using the formula found on page 12-11.
RXD	I/O	Receive Serial Data In modes 1, 2, and 3, RXD receives serial port input data. In mode 0, it functions as either an input or an open-drain output for data. RXD is multiplexed with P2.1.
T1CLK	I	Timer 1 External Clock External clock for timer 1. Timer 1 increments (or decrements) on both rising and falling edges of T1CLK. Also used in conjunction with T1DIR for quadrature counting mode. and External clock for the serial I/O baud-rate generator input (program selectable). T1CLK is multiplexed with P1.4.
T2CLK	I	Timer 2 External Clock External clock for timer 2. Timer 2 increments (or decrements) on both rising and falling edges of T2CLK. Also used in conjunction with T2DIR for quadrature counting mode. T2CLK is multiplexed with P1.6.
T1DIR	I	Timer 1 External Direction External direction (up/down) for timer 1. Timer 1 increments when T1DIR is high and decrements when it is low. Also used in conjunction with T1CLK for quadrature counting mode. T1DIR is multiplexed with P1.5.
T2DIR	I	Timer 2 External Direction External direction (up/down) for timer 2. Timer 2 increments when T2DIR is high and decrements when it is low. Also used in conjunction with T2CLK for quadrature counting mode. T2DIR is multiplexed with P1.7.
TXD	O	Transmit Serial Data In serial I/O modes 1, 2, and 3, TXD transmits serial port output data. In mode 0, it is the serial clock output. TXD is multiplexed with P2.0.

Table B-3. Signal Descriptions (Continued)

Name	Type	Description
V_{CC}	PWR	Digital Supply Voltage Connect each V_{CC} pin to the digital supply voltage.
V_{SS}	GND	Digital Circuit Ground Connect each V_{SS} pin to ground through the lowest possible impedance path.
WR#	O	Write† This active-low output indicates that an external write is occurring. This signal is asserted only during external memory writes. WR# is multiplexed with WRL#. † The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.
WRH#	O	Write High† During 16-bit bus cycles, this active-low output signal is asserted for high-byte writes and word writes to external memory. During 8-bit bus cycles, WRH# is asserted for all write operations. WRH# is multiplexed with BHE#. † The chip configuration register 0 (CCR0) determines whether this pin functions as BHE# or WRH#. CCR0.2 = 1 selects BHE#; CCR0.2 = 0 selects WRH#.
WRL#	O	Write Low† During 16-bit bus cycles, this active-low output signal is asserted for low-byte writes and word writes. During 8-bit bus cycles, WRL# is asserted for all write operations. WRL# is multiplexed with WR#. † The chip configuration register 0 (CCR0) determines whether this pin functions as WR# or WRL#. CCR0.2 = 1 selects WR#; CCR0.2 = 0 selects WRL#.
XTAL1	I	Input Crystal/Resonator or External Clock Input Input to the on-chip oscillator, internal phase-locked loop circuitry (80C196NU), and the internal clock generators. The internal clock generators provide the peripheral clocks, CPU clock, and CLKOUT signal. When using an external clock source instead of the on-chip oscillator, connect the clock input to XTAL1. The external clock signal must meet the V_{IH} specification for XTAL1 (see datasheet).
XTAL2	O	Inverted Output for the Crystal/Resonator Output of the on-chip oscillator inverter. Leave XTAL2 floating when the design uses a external clock source instead of the on-chip oscillator.

B.3 DEFAULT CONDITIONS

Table B-5 lists the default functions of the I/O and control pins of the 8XC196NP and 80C196NU with their values during various operating conditions. Table B-4 defines the symbols used to represent the pin status. Refer to the DC Characteristics table in the datasheet for actual specifications for V_{OL}, V_{IL}, V_{OH}, and V_{IH}.

Table B-4. Definition of Status Symbols

Symbol	Definition
0	Voltage less than or equal to V_{OL}, V_{IL}
1	Voltage greater than or equal to V_{OH}, V_{IH}
HiZ	High impedance
LoZ0	Low impedance; strongly driven low
LoZ1	Low impedance; strongly driven high

Symbol	Definition
MD0	Medium pull-down
MD1	Medium pull-up
WK0	Weak pull-down
WK1	Weak pull-up
ODIO	Open-drain I/O

Table B-5. 8XC196NP and 80C196NU Pin Status

Port Pins	Multiplexed With	During RESET# Active	Upon RESET# Inactive (Note 11)	Idle	Power-down (NP/NU) and Standby (NU only)	Hold	Bus Idle
P1.3:0	EPA3:0	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P1.4	T1CLK	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P1.5	T1DIR	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P1.6	T2CLK	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P1.7	T2DIR	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P2.0	TXD	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P2.1	RXD	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P2.2	EXTINT0	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P2.3	BREQ#	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P2.4	EXTINT1	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P2.5	HOLD#	WK1	WK1	(Note 1)	(Note 1)	Force 0	—
P2.6	HLDA#	WK1	WK1	(Note 1)	(Note 1)	0	—
P2.7	CLKOUT	CLKOUT active; LoZ0/1	CLKOUT active; LoZ0/1	(Note 1)	(Note 2)	(Note 1)	—
P3.0	CS0#	WK1	1 (NP only) 0 (NU only)	(Note 3)	(Note 3)	(Note 4)	—
P3.5:1	CS5:1#	WK1	WK1	(Note 3)	(Note 3)	(Note 4)	—
P3.6	EXTINT2	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P3.7	EXTINT3	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
P4.2:0	PWM2:0	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—

Table B-5. 8XC196NP and 80C196NU Pin Status (Continued)

Port Pins	Multiplexed With	During RESET# Active	Upon RESET# Inactive (Note 11)	Idle	Power-down (NP/NU) and Standby (NU only)	Hold	Bus Idle
P4.3	—	WK1	WK1	(Note 1)	(Note 1)	(Note 1)	—
EPORT.3:0	A19:16	WK1	1	(Note 5)	(Note 5)	(Note 6)	(Note 8)
—	A15:0	WK1	LoZ0	(Note 7)	(Note 7)	HiZ	LoZ0
—	AD15:0	WK1	LoZ0	(Note 7)	(Note 7)	HiZ	LoZ0
—	ALE	WK0	0	(Note 9)	(Note 9)	WK0	LoZ0
—	BHE#	WK1	1	(Note 10)	(Note 10)	WK1	LoZ1
—	EA# (NP only)	HiZ	HiZ	HiZ	HiZ	HiZ	—
—	INST	WK0	0	(Note 9)	(Note 9)	WK0	LoZ0
—	NMI	WK0	WK0	WK0	WK0	WK0	—
—	ONCE	MD0	MD0	MD0	MD0	MD0	—
—	PLLEN1 (NU only)	HiZ	HiZ	HiZ	HiZ	HiZ	—
—	PLLEN2 (NU only)	MD0	MD0	MD0	MD0	MD0	—
—	RD#	WK1	1	(Note 10)	(Note 10)	WK1	LoZ1
—	READY	WK1	WK1	WK1	WK1	WK1	—
—	RESET#	0	WK1	WK1	WK1	WK1	—
—	RPD	LoZ1	LoZ1	LoZ1	LoZ1	LoZ1	—
—	WR#	WK1	1	(Note 10)	(Note 10)	WK1	LoZ1
XTAL1	—	Osc input, HiZ	Osc input, HiZ	Osc input, HiZ	Osc input, HiZ	Osc input, HiZ	—
XTAL2	—	Osc output, LoZ0/1	Osc output, LoZ0/1	Osc output, LoZ0/1	HiZ	Osc output, LoZ0/1	—

NOTE:
1. If P*x*_MODE.*y* = 0, then port is as programmed. If P*x*_MODE.*y* = 1, then as specified by the associated peripheral.
2. If P2_MODE.7 = 0, then port is as programmed. If P2_MODE.7 = 1, then 1.
3. Used as chip select: If HLDA# = 0, then WK1. If HLDA# = 1, then LoZ1. Used as port: then port is as programmed.
4. Used as chip select: WK1. Used as port: then port is as programmed.
5. When used as extended address: If HLDA# = 1, then 0. If HLDA# = 0, then HiZ
   When used as EPORT, then port value.
6. When used as extended address, then HiZ. When used as EPORT, then port value.
7. If HLDA# = 1, then LoZ0. If HLDA# = 0, then HiZ.
8. When used as extended address: then previous address. When used as EPORT: then port value.
9. If HLDA# = 1, then LoZ0. If HLDA# = 0, then WK0.
10. If HLDA# = 1, then LoZ1. If HLDA# = 0, then WK1.
11. The values in this column are valid until user code configures the specific signal (i.e., until P*x*_MODE is written).

C

Registers

APPENDIX C
REGISTERS

This appendix provides reference information about the device registers. Table C-1 lists the modules and major components of the device with their related configuration and status registers. Table C-2 lists the registers, arranged alphabetically by mnemonic, along with their names, addresses, and reset values. Following the tables, individual descriptions of the registers are arranged alphabetically by mnemonic.

Table C-1. Modules and Related Registers

Chip Configuration	**Chip-select Units (*x* = 0–5)**	**CPU (*x* = 0, 2)**	**EPA (*x* = 0–3)**
CCR0	ADDRCOM*x*	ACC_0*x* (80C196NU)	EPA_MASK
CCR1	ADDRMSK*x*	ACC_STAT (80C196NU)	EPA_PEND
	BUSCON*x*	ONES_REG	EPA*x*_CON
		PSW	EPA*x*_TIME
		SP	
		ZERO_REG	
Extended Port	**I/O Ports (*x* = 1–4)**	**Interrupts**	**Memory Control**
EP_DIR	P*x*_DIR	INT_MASK	WSR
EP_MODE	P*x*_MODE	INT_MASK1	WSR1 (80C196NU)
EP_PIN	P*x*_PIN	INT_PEND	
EP_REG	P*x*_REG	INT_PEND1	
PWM (*x* = 0–2)	**PTS**	**Serial Port**	**Timers (*x* = 1–2)**
CON_REG0	PTSSEL	SBUF_RX	TIMER*x*
PWM*x*_CONTROL	PTSSRV	SBUF_TX	T*x*CONTROL
		SP_BAUD	
		SP_CON	
		SP_STATUS	

Table C-2. Register Name, Address, and Reset Status

Register Mnemonic	Register Name	Hex Address	Binary Reset Value	
			High	Low
ACC_00 (NU)	Accumulator 0	000CH	0000 0000	0000 0000
ACC_02 (NU)	Accumulator 2	000EH	0000 0000	0000 0000
ACC_STAT (NU)	Accumulator Control and Status	000BH		0000 0000
ADDRCOM0	Address Compare 0	1F40H	0000 1111	0010 0000
ADDRCOM1	Address Compare 1	1F48H	XXXX 0000	0000 0000
ADDRCOM2	Address Compare 2	1F50H	XXXX 0000	0000 0000
ADDRCOM3	Address Compare 3	1F58H	XXXX 0000	0000 0000
ADDRCOM4	Address Compare 4	1F60H	XXXX 0000	0000 0000
ADDRCOM5	Address Compare 5	1F68H	XXXX 0000	0000 0000
ADDRMSK0	Address Mask 0	1F42H	XXXX 1111	1111 1111
ADDRMSK1	Address Mask 1	1F4AH	XXXX 1111	1111 1111
ADDRMSK2	Address Mask 2	1F52H	XXXX 1111	1111 1111
ADDRMSK3	Address Mask 3	1F5AH	XXXX 1111	1111 1111
ADDRMSK4	Address Mask 4	1F62H	XXXX 1111	1111 1111
ADDRMSK5	Address Mask 5	1F6AH	XXXX 1111	1111 1111
BUSCON0	Bus Control 0	1F44H		0000 0011
BUSCON1	Bus Control 1	1F4CH		0000 0000
BUSCON2	Bus Control 2	1F54H		0000 0000
BUSCON3	Bus Control 3	1F5CH		0000 0000
BUSCON4	Bus Control 4	1F64H		0000 0000
BUSCON5	Bus Control 5	1F6CH		0000 0000
CCR0	Chip Configuration 0	FF2018H		XXXX XXXX
CCR1	Chip Configuration 1	FF201AH		XXXX XXXX
CON_REG0	PWM Clock Prescaler Control 0	1FB6H		1111 1110
EP_DIR	Extended Port I/O Direction	1FE3H		1111 1111
EP_MODE	Extended Port Mode	1FE1H		1111 1111
EP_PIN	Extended Port Pin Input	1FE7H		XXXX XXXX
EP_REG	Extended Port Data Output	1FE5H		XXXX 0000
EPA_MASK	EPA Mask	1F9CH		1010 1010
EPA_PEND	EPA Pending	1F9EH		1010 1010
EPA0_CON	EPA Capture/Comp 0 Control	1F80H		0000 0000
EPA1_CON	EPA Capture/Comp 1 Control	1F84H	0000 0000	0000 0000

Table C-2. Register Name, Address, and Reset Status (Continued)

Register Mnemonic	Register Name	Hex Address	Binary Reset Value	
			High	Low
EPA2_CON	EPA Capture/Comp 2 Control	1F88H	0000	0000
EPA3_CON	EPA Capture/Comp 3 Control	1F8CH	0000 0000	0000 0000
EPA0_TIME	EPA Capture/Comp 0 Time	1F82H	0000 0000	0000 0000
EPA1_TIME	EPA Capture/Comp 1 Time	1F86H	0000 0000	0000 0000
EPA2_TIME	EPA Capture/Comp 2 Time	1F8AH	0000 0000	0000 0000
EPA3_TIME	EPA Capture/Comp 3 Time	1F8EH	0000 0000	0000 0000
INT_MASK	Interrupt Mask	0008H		0000 0000
INT_MASK1	Interrupt Mask 1	0013H		0000 0000
INT_PEND	Interrupt Pending	0009H		0000 0000
INT_PEND1	Interrupt Pending 1	0012H		0000 0000
ONES_REG	Ones Register	0002H	1111 1111	1111 1111
P1_DIR	Port 1 I/O Direction	1FD2H		1111 1111
P1_MODE	Port 1 Mode	1FD0H		0000 0000
P1_PIN	Port 1 Pin Input	1FD6H		XXXX XXXX
P1_REG	Port 1 Data Output	1FD4H		1111 1111
P2_DIR	Port 2 I/O Direction	1FD3H		1111 1111
P2_MODE	Port 2 Mode	1FD1H		1000 0000
P2_PIN	Port 2 Pin Input	1FD7H		XXXX XXXX
P2_REG	Port 2 Data Output	1FD5H		1111 1111
P3_DIR	Port 3 I/O Direction	1FDAH		1111 1111
P3_MODE	Port 3 Mode	1FD8H		0000 0001
P3_PIN	Port 3 Pin Input	1FDEH		XXXX XXXX
P3_REG	Port 3 Data Output	1FDCH		1111 1111
P4_DIR	Port 4 I/O Direction	1FDBH		1111 1111
P4_MODE	Port 4 Mode	1FD9H		0000 0000
P4_PIN	Port 4 Pin Input	1FDFH		XXXX XXXX
P4_REG	Port 4 Data Output	1FDDH		1111 1111
PSW	Program Status Word			
PTSSEL	PTS Select	0004H	0000 0000	0000 0000
PTSSRV	PTS Service	0006H	0000 0000	0000 0000
PWM0_CONTROL	PWM 0 Control	1FB0H		0000 0000
PWM1_CONTROL	PWM 1 Control	1FB2H		0000 0000
PWM2_CONTROL	PWM 2 Control	1FB4H		0000 0000
SBUF_RX	Serial Port Receive Buffer	1FB8H		0000 0000

Table C-2. Register Name, Address, and Reset Status (Continued)

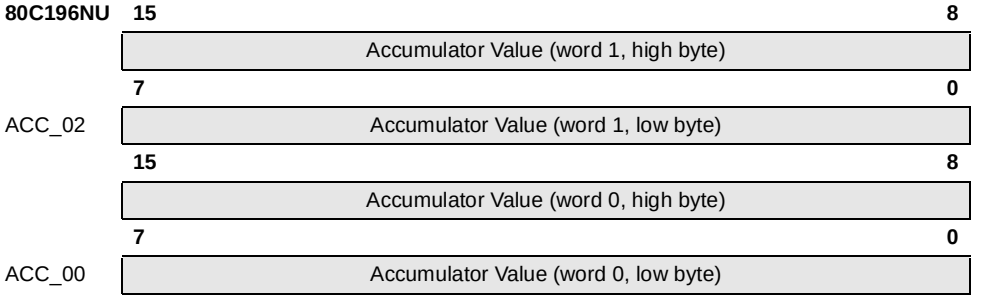
Register Mnemonic	Register Name	Hex Address	Binary Reset Value	
			High	Low
SBUF_TX	Serial Port Transmit Buffer	1FBAH	0000 0000	0000 0000
SP	Stack Pointer	0018H	XXXX XXXX	XXXX XXXX
SP_BAUD	Serial Port Baud Rate	1FBCH	0000 0000	0000 0000
SP_CON	Serial Port Control	1FBBH	0000 0000	0000 0000
SP_STATUS	Serial Port Status	1FB9H	0000 0000	0000 1011
T1CONTROL	Timer 1 Control	1F90H	0000 0000	0000 0000
T2CONTROL	Timer 2 Control	1F94H	0000 0000	0000 0000
TIMER1	Timer 1 Value	1F92H	0000 0000	0000 0000
TIMER2	Timer 2 Value	1F96H	0000 0000	0000 0000
WSR	Window Selection	0014H	0000 0000	0000 0000
WSR1 (NU)	Window Selection 1	0015H	0000 0000	0000 0000
ZERO_REG	Zero Register	0000H	0000 0000	0000 0000

ACC_0x

ACC_0*x*
***x* = 0, 2 (80C196NU)**

Address: Table C-3
Reset State:

The 32-bit accumulator register (ACC_0*x*) resides at locations 0C–0FH. You can read from or write to the accumulator register as two words at locations 0CH and 0EH.



Bit Number	Function
15:0	Accumulator Value You can read this register to determine the current value of the accumulator. You can write to this register to clear or preload a value into the accumulator.

Table C-3. ACC_0*x* Addresses and Reset Values

Register	Address	Reset Value
ACC_00	000CH	00H
ACC_02	000EH	00H

ACC_STAT

ACC_STAT
(80C196NU)

Address: 0BH
Reset State: 00H

The accumulator control and status (ACC_STAT) register enables and disables fractional and saturation modes and contains three status flags that indicate the status of the accumulator's contents.

	7							0
80C196NU	FME	SME	—	—	—	STOVF	OVF	STSAT

Bit Number	Bit Mnemonic	Function
7	FME	Fractional Mode Enable Set this bit to enable fractional mode. (See Table C-4.) In this mode, the result of a signed multiplication instruction is shifted left by one bit before it is added to the contents of the accumulator. For unsigned multiplication, this bit is ignored.
6	SME	Saturation Mode Enable Set this bit to enable saturation mode. (See Table C-4.) In this mode, the result of a signed multiplication operation is **not** allowed to overflow or underflow. For unsigned multiplication, this bit is ignored.
5:3	—	Reserved; for compatibility with future devices, write zeros to these bits.
2	STOVF	Sticky Overflow Flag For unsigned multiplication, this bit is set if a carry out of bit 31 occurs. Unless saturation mode is enabled, this bit is set for signed multiplication to indicate that the sign bit of the accumulator and the sign bit of the addend are equal, but the sign bit of the result is the opposite. (See Table C-4.) Software can clear this flag; hardware does **not** clear it.
1	OVF	Overflow Flag This bit indicates that an overflow occurred during the preceding accumulation. (See Table C-4.) This flag is dynamic; it can change after each accumulation.
0	STSAT	Sticky Saturation Flag This bit indicates that a saturation has occurred during accumulation with saturation mode enabled. (See Table C-4.) Software can clear this flag; hardware does **not** clear it.

Table C-4. Effect of SME and FME Bit Combinations

SME	FME	Description
0	0	Sets the OVF and STOVF flags if the sign bits of the accumulator and the addend (the number to be added to the contents of the accumulator) are equal, but the sign bit of the result is the opposite.
0	1	Shifts the addend (the number to be added to the contents of the accumulator) left by one bit before adding it to the accumulator. Sets the OVF and STOVF flags if the sign bits of the accumulator and the addend are equal, but the sign bit of the result is the opposite.
1	0	Accumulates a signed integer value up or down to saturation and sets the STSAT flag. Positive saturation changes the accumulator value to 7FFFFFFFH; negative saturation changes the accumulator value to 80000000H. Accumulation proceeds normally after saturation, which means that the accumulator value can increase from a negative saturation or decrease from a positive saturation.
1	1	Shifts the addend (the number to be added to the contents of the accumulator) left by one bit before adding it to the accumulator. Accumulates a signed integer value up or down to saturation and sets the STSAT flag. Positive saturation changes the accumulator value to 7FFFFFFFH; negative saturation changes the accumulator value to 80000000H. Accumulation proceeds normally after saturation, which means that the accumulator value can increase from a negative saturation or decrease from a positive saturation.

ADDRCOMx

ADDRCOM*x*
***x* = 0–5**

Address: Table C-5
Reset State:

The address compare (ADDRCOM*x*) register specifies the base (lowest) address of the address range. The base address of a 2^n-byte address range must be on a 2^n-byte boundary.

15							8
—	—	—	—	BASE19	BASE18	BASE17	BASE16
7							**0**
BASE15	BASE14	BASE13	BASE12	BASE11	BASE10	BASE9	BASE8

Bit Number	Bit Mnemonic	Function
15:12	—	Reserved; for compatibility with future devices, write zeros to these bits.
11:0	BASE19:8	Base Address Bits These bits are the 12 most-significant bits of the base address of the address range assigned to chip-select *x*.

Table C-5. ADDRCOM*x* Addresses and Reset Values

Register	Address	Reset Value
ADDRCOM0	1F40H	0F20H
ADDRCOM1	1F48H	X000H
ADDRCOM2	1F50H	X000H
ADDRCOM3	1F58H	X000H
ADDRCOM4	1F60H	X000H
ADDRCOM5	1F68H	X000H

ADDRMSKx

ADDRMSK*x*
***x* = 0–5**

Address: Table C-6
Reset State:

The address mask (ADDRMSK*x*) register, together with the address compare register, defines the address range that is assigned to the chip-select *x* output, CS*x*#. The address mask register determines the size of the address range, which must be 2^n bytes, where n = 8, 9, . . , 20. For a 2^n-byte address range, calculate $n_1 = 20 - n$, and set the n_1 most-significant bits of MASK19:8 in the address mask register.

15							8
—	—	—	—	MASK19	MASK18	MASK17	MASK16
7							**0**
MASK15	MASK14	MASK13	MASK12	MASK11	MASK10	MASK9	MASK8

Bit Number	Bit Mnemonic	Function
15:12	—	Reserved; for compatibility with future devices, write zeros to these bits.
11:0	MASK19:8	Address Mask Bits For a 2^n-byte address range, set the n_1 most-significant bits of MASK19:8, where $n_1 = 20 - n$.

Table C-6. ADDRMSK*x* Addresses and Reset Values

Register	Address	Reset Value
ADDRMSK0	1F42H	XFFFH
ADDRMSK1	1F4AH	XFFFH
ADDRMSK2	1F52H	XFFFH
ADDRMSK3	1F5AH	XFFFH
ADDRMSK4	1F62H	XFFFH
ADDRMSK5	1F6AH	XFFFH

BUSCONx

BUSCON*x*
***x* = 0–5**

Address: Table C-7
Reset State:

For the address range assigned to chip-select *x*, the bus control (BUSCON*x*) register specifies the number of wait states, the bus width, and the address/data multiplexing for all external bus cycles that access address range *x*.

7							0
DEMUX	BW16	—	—	—	—	WS1	WS0

Bit Number	Bit Mnemonic	Function
7	DEMUX	Address/Data Multiplexing This bit specifies the address/data multiplexing on AD15:0 for all external accesses to the address range assigned to chip-select output *x*. 0 = multiplexed 1 = demultiplexed
6	BW16	Bus Width This bit specifies the bus width for all external accesses to the address range assigned to chip-select output *x*. 0 = 8 bits 1 = 16 bits
5:2	—	Reserved; for compatibility with future devices, write zeros to these bits.
1:0	WS1:0	Wait States These bits specify the number of wait states for all external accesses to the address range assigned to chip-select output *x*. **WS1 WS0 Wait States** 0 0 0 0 1 1 1 0 2 1 1 3

Table C-7. BUSCON*x* Addresses and Reset Values

Register	Address	Reset Value
BUSCON0	1F44H	03H
BUSCON1	1F4CH	00H
BUSCON2	1F54H	00H
BUSCON3	1F5CH	00H
BUSCON4	1F64H	00H
BUSCON5	1F6CH	00H

CCR0

CCR0 no direct access†

The chip configuration 0 (CCR0) register enables or disables powerdown and standby (80C196NU only) modes and selects the write-control mode. It also contains the bus-control parameters for fetching chip configuration byte 1.

7							0
1	1	WS1	WS0	DEMUX	BHE#	BW16	PD

Bit Number	Bit Mnemonic	Function
7:6	1	To guarantee device operation, write ones to these bits.
5:4	WS1:0	Wait States These two bits control the number of wait states that are used for an external fetch of CCB1. **WS0 WS1** 0 0 zero wait states 0 1 one wait state 1 0 two wait states 1 1 three wait states
3	DEMUX	Select Demultiplexed Bus Selects the demultiplexed bus mode for an external fetch of CCB1: 0 = multiplexed — address and data are multiplexed on AD15:0. 1 = demultiplexed — data only on AD15:0.
2	BHE#	Write-control Mode Selects the write-control mode, which determines the functions of the BHE#/WRH# and WR#/WRL# pins for external bus cycles: 0 = write strobe mode: the BHE#/WRH# pin operates as WRH#, and the WR#/WRL# pin operates as WRL#. 1 = standard write-control mode: the BHE#/WRH# pin operates as BHE#, and the WR#/WRL# pin operates as WR#.
1	BW16	Buswidth Control Selects the bus width for an external fetch of CCB1: 0 = 8-bit bus 1 = 16-bit bus
0	PD	Powerdown Enable Enables or disables the IDLPD #2 and IDLPD #3 instructions. When enabled, the IDLPD #2 instruction causes the microcontroller to enter powerdown mode and for the 80C196NU only, the IDLPD #3 instruction causes the microcontroller to enter standby mode. 0 = disable powerdown and standby modes 1 = enable powerdown and standby modes If your design uses powerdown or standby mode, set this bit when you program the CCBs. If it does not, clearing this bit when you program the CCBs will prevent accidental entry into powerdown and standby mode†. (Chapter 12, “Special Operating Modes,” discusses powerdown and standby modes.)

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

CCR1

CCR1 no direct access†

The chip configuration 1 (CCR1) register selects the 16-bit or 24-bit addressing mode and (for the 8XC196NP only) controls whether the internal ROM is mapped into two address ranges, FF2000–FF2FFFH and 002000–002FFFH, or into FF2000–FF2FFFH only.

	7							0
8XC196NP	1	1	0	1	1	REMAP	MODE64	—
	7							**0**
80C196NU	1	1	DM	1	1	—	MODE64	—

Bit Number	Bit Mnemonic	Function
7:6	1	To guarantee device operation, write ones to these bits.
5††	DM	Deferred Mode Enables the deferred bus-cycle mode. If the 80C196NU is using a demultiplexed bus and deferred mode is enabled, a delay of 2t occurs in the first bus cycle following a chip-select output change and the first write cycle following a read cycle. (See "Deferred Bus-cycle Mode (80C196NU Only)" on page 13-40.) 0 = deferred bus-cycle mode disabled 1 = deferred bus-cycle mode enabled
4:3	1	To guarantee device operation, write ones to these bits.
2††	REMAP	Internal ROM Mapping Controls the internal ROM mapping. 0 = ROM maps to FF2000–FF2FFFH only 1 = ROM maps to FF2000–FF2FFFH and 002000–002FFFH
1	MODE64	Addressing Mode Selects 64-Kbyte or 1-Mbyte addressing. 0 = selects 1-Mbyte addressing 1 = selects 64-Kbyte addressing
0	—	Reserved; for compatibility with future devices, write zero to this bit.

† The CCRs are loaded with the contents of the chip configuration bytes (CCBs) after a device reset. The CCBs reside in nonvolatile memory at addresses FF2018H (CCB0) and FF201AH (CCB1).

†† Bit 5 is reserved on the 8XC196NP device and bit 2 is reserved on the 80C196NU device. For compatibility with future devices, write zeros to these bits.

CON_REG0

CON_REG0

Address: 1FB6H
Reset State: FEH

The control (CON_REG0) register controls the clock prescaler for the three pulse-width modulators (PWM0–PWM2).

	7							0
8XC196NP	—	—	—	—	—	—	—	CLK0†
	7							**0**
80C196NU	—	—	—	—	—	—	CLK1	CLK0

Bit Number	Bit Mnemonic	Function
7:1 (NP) 7:2 (NU)	—	Reserved; for compatibility with future devices, write zeros to these bits.
0 (NP)	CLK0	Enable PWM Clock Prescaler This bit controls the PWM output period by enabling or disabling the clock prescaler (divide-by-two) on the three pulse-width modulators (PWM0–PWM2). 0 = disable; PWM output period is 512 state times 1 = enable; PWM output period is 1024 state times
1:0 (NU)	CLK1:0	Enable PWM Clock Prescaler These bits control the PWM output period on the three pulse-width modulators (PWM0–PWM2). **CLK1 CLK0** 0 0 disable clock prescaler 0 1 enable divide-by-two prescaler; PWM output period is 1024 state times 1 X enable divide-by-four prescaler; PWM output period is 2048 state times

† This bit was called SLOW_PWM in earlier documentation for the 8XC196NP.

EP_DIR

EP_DIR Address: 1FE3H
Reset State: FFH

In I/O mode, each bit of the extended port I/O direction (EP_DIR) register controls the direction of the corresponding pin. Clearing a bit configures a pin as a complementary output; setting a bit configures a pin as either an input or an open-drain output. (Open-drain outputs require external pull-ups).

Any pin that is configured for its extended-address function is forced to the complementary output mode except during reset, hold, idle, powerdown, and standby. (Standby mode is available only on the 80C196NU.)

7							0
—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:4	—	Reserved; always write as ones.
3:0	PIN3:0	Extended Address Port Pin *x* Direction This bit configures EPORT.*x* as a complementary output or an input/open-drain output. 0 = complementary output 1 = input or an open-drain output

EP_MODE

EP_MODE

Address: 1FE1H
Reset State: FFH

Each bit of the extended port mode (EP_MODE) register controls whether the corresponding pin functions as a standard I/O port pin or as an extended-address signal. Setting a bit configures a pin as an extended-address signal; clearing a bit configures a pin as a standard I/O port pin.

7							0
—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:4	—	Reserved; always write as zeros.
3:0	PIN3:0	Extended Address Port Pin *x* Mode This bit determines the mode of EPORT.*x*: 0 = standard I/O port pin 1 = extended-address signal

EP_PIN

EP_PIN Address: 1FE7H
Reset State: XXH

Each bit of the extended port input (EP_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

7							0
—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:4	—	Reserved; always write as zeros.
3:0	PIN3:0	Extended Address Port Pin *x* Input This bit contains the current state of EPORT.*x*.

EP_REG

EP_REG

Address: 1FE5H
Reset State: X0H

Each bit of the extended port data output (EP_REG) register contains data to be driven out by the corresponding pin. When a pin is configured as standard I/O (EP_MODE.*x* = 0), the result of a CPU write to EP_REG is immediately visible on the pin.

During nonextended data accesses, EP_REG contains the value of the memory page that is to be accessed. For compatibility with software tools, clear the EP_REG bit for any EPORT pin that is configured as an extended-address signal (EP_MODE.*x* set).

80C196NU Only: For nonextended data accesses, the 80C196NU forces the page address to 00H. You cannot change pages by modifying EP_REG.

7							0
—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:4	—	Reserved; always write as zeros.
3:0	PIN3:0	Extended Address Port Pin *x* Output If EPORT.*x* is to be used as an output, write the data that it is to drive out. If EPORT.*x* is to be used as an input, set this bit. For the 8XC196NP, if EPORT.*x* is to be used as an address line, write the correct value for the memory page to be accessed by nonextended instructions. The 80C196NU forces the page address to 00H. You cannot change pages by modifying EP_REG

EPA_MASK

EPA_MASK

Address: 1F9CH
Reset State: AAH

The EPA interrupt mask (EPA_MASK) register enables or disables (masks) the multiplexed EPA3:0 overrun interrupts (OVR3:0).

7 **0**

—	OVR3	—	OVR2	—	OVR1	—	OVR0

Bit Number	Bit Mnemonic	Function
7, 5, 3, 1	—	Reserved; for compatibility with future devices, write zeros to these bits.
6, 4, 2, 0	OVR3 OVR2 OVR1 OVR0	Setting this bit enables the corresponding source as a shared overrun interrupt source. The shared overrun interrupts (OVR0_1 and OVR2_3) are enabled by setting their interrupt enable bits in the interrupt mask 1 (INT_MASK1) register.

EPA_PEND

EPA_PEND

Address: 1F9EH
Reset State: AAH

When hardware detects a pending EPA3:0 overrun interrupt (OVR3:0), it sets the corresponding bit in the EPA interrupt pending (EPA_PEND) register. OVR0 and OVR1 are multiplexed to share one bit (OVR0_1) in the INT_PEND1 register. Similarly, OVR2 and OVR3 are multiplexed to share another bit (OVR2_3) in the INT_PEND1 register.

7 **0**

—	OVR3	—	OVR2	—	OVR1	—	OVR0

Bit Number	Function
7, 5, 3, 1	Reserved. These bits are undefined.
6, 4, 2, 0	Any set bit indicates that the corresponding overrun interrupt source is pending.

NOTE: This register was called EPA_STAT in previous documentation for the 8XC196NP.

EPAx_CON

EPA*x*_CON
***x* = 0–3**

Address: Table C-8
Reset State:

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels. The registers for EPA0 andEPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit. This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON must be addressed as words, while the others can be addressed as bytes.

***x* = 1, 3**

15							8
—	—	—	—	—	—	—	RM
7							**0**
TB	CE	M1	M0	RE	—	ROT	ON/RT

***x* = 0, 2**

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

Bit Number	Bit Mnemonic	Function
15:9†	—	Reserved; always write as zeros.
8†	RM	Remap Feature The remap feature applies to the compare mode of the EPA1 and EPA3 only. When the remap feature of EPA1 is enabled, EPA capture/compare channel 0 shares output pin EPA1 with EPA capture/compare channel 1. When the remap feature of EPA3 is enabled, EPA capture/compare channel 2 shares output pin EPA3 with EPA capture/compare channel 3. 0 = remap feature disabled 1 = remap feature enabled
7	TB	Time Base Select Specifies the reference timer. 0 = timer 1 is the reference timer and timer 2 is the opposite timer 1 = timer 2 is the reference timer and timer 1 is the opposite timer A compare event (clearing, setting, or toggling an output pin; and/or resetting either timer) occurs when the reference timer matches the time programmed in the event-time register. When a capture event (falling edge, rising edge, or an edge change on the EPA*x* pin) occurs, the reference timer value is saved in the EPA event-time register (EPA*x*_TIME).
6	CE	Compare Enable Determines whether the EPA channel operates in capture or compare mode. 0 = capture mode 1 = compare mode

† These bits apply to the EPA1_CON and EPA3_CON registers only.

EPA*x*_CON (Continued)
***x* = 0–3**

Address: Table C-8
Reset State:

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels. The registers for EPA0 andEPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit. This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON must be addressed as words, while the others can be addressed as bytes.

***x* = 1, 3**

15							8
—	—	—	—	—	—	—	RM

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

***x* = 0, 2**

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

Bit Number	Bit Mnemonic	Function
5:4	M1:0	EPA Mode Select In capture mode, specifies the type of event that triggers an input capture. In compare mode, specifies the action that the EPA executes when the reference timer matches the event time. **M1 M0 Capture Mode Event** 0 0 no capture 0 1 capture on falling edge 1 0 capture on rising edge 1 1 capture on either edge **M1 M0 Compare Mode Action** 0 0 no output 0 1 clear output pin 1 0 set output pin 1 1 toggle output pin
3	RE	Re-enable Re-enable applies to the compare mode only. It allows a compare event to continue to execute each time the event-time register (EPA*x*_TIME) matches the reference timer rather than only upon the first time match. 0 = compare function is disabled after a single event 1 = compare function always enabled
2	—	Reserved; always write as zero.

† These bits apply to the EPA1_CON and EPA3_CON registers only.

EPAx_CON

EPA*x*_CON (Continued)
***x* = 0–3**

Address: Table C-8
Reset State:

The EPA control (EPA*x*_CON) registers control the functions of their assigned capture/compare channels. The registers for EPA0 andEPA2 are identical. The registers for EPA1 and EPA3 have an additional bit, the remap bit. This added bit (bit 8) requires an additional byte, so EPA1_CON and EPA3_CON must be addressed as words, while the others can be addressed as bytes.

***x* = 1, 3**

15							8
—	—	—	—	—	—	—	RM
7							**0**
TB	CE	M1	M0	RE	—	ROT	ON/RT

***x* = 0, 2**

7							0
TB	CE	M1	M0	RE	—	ROT	ON/RT

Bit Number	Bit Mnemonic	Function
1	ROT	Reset Opposite Timer Controls different functions for capture and compare modes. **In Capture Mode:** 0 = causes no action 1 = resets the opposite timer **In Compare Mode:** Selects the timer that is to be reset if the RT bit is set. 0 = selects the reference timer for possible reset 1 = selects the opposite timer for possible reset The TB bit (bit 7) selects which is the reference timer and which is the opposite timer.
0	ON/RT	Overwrite New/Reset Timer The ON/RT bit functions as overwrite new in capture mode and reset timer in compare mode. **In Capture Mode (ON):** An overrun error is generated when an input capture occurs while the event-time register (EPA*x*_TIME) and its buffer are both full. When an overrun occurs, the ON bit determines whether old data is overwritten or new data is ignored: 0 = ignores new data 1 = overwrites old data in the buffer **In Compare Mode (RT):** 0 = disables the reset function 1 = resets the ROT-selected timer

† These bits apply to the EPA1_CON and EPA3_CON registers only.

Table C-8. EPA*x*_CON Addresses and Reset Values

Register	Address	Reset Value
EPA0_CON	1F80H	00H
EPA1_CON	1F84H	0000H
EPA2_CON	1F88H	00H
EPA3_CON	1F8CH	0000H

EPAx_TIME

EPA*x*_TIME
***x* = 0–3**

Address: Table C-9
Reset State:

The EPA time (EPA*x*_TIME) registers are the event-time registers for the EPA channels. In capture mode, the value of the reference timer is captured in EPA*x*_TIME when an input transition occurs. Each event-time register is buffered, allowing the storage of two capture events at once. In compare mode, the EPA triggers a compare event when the reference timer matches the value in EPA*x*_TIME. EPA*x*_TIME is not buffered for compare mode.

15	8
EPA Timer Value (high byte)	
7	**0**
EPA Timer Value (low byte)	

Bit Number	Function
15:0	EPA Time Value When an EPA channel is configured for capture mode, this register contains the value of the reference timer when the specified event occurred. When an EPA channel is configured for compare mode, write the compare event time to this register.

Table C-9. EPA*x*_TIME Addresses and Reset Values

Register	Address	Reset Value
EPA0_TIME	1F82H	0000H
EPA1_TIME	1F86H	0000H
EPA2_TIME	1F8AH	0000H
EPA3_TIME	1F8EH	0000H

INT_MASK

INT_MASK

Address: 0008H
Reset State: 00H

The interrupt mask (INT_MASK) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK is the low byte of the processor status word (PSW); therefore, PUSHF or PUSHA saves this register on the stack and POPF or POPA restores it.

7 **0**

EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM2	OVRTM1

Bit Number	Function
7:3 1:0	Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows: **Bit Mnemonic** — **Interrupt** — **Standard Vector** EPA0 — EPA Capture/Compare Channel 0 — FF200EH RI — SIO Receive — FF200CH TI — SIO Transmit — FF200AH EXTINT1 — EXTINT1 pin — FF2008H EXTINT0 — EXTINT0 pin — FF2006H OVRTM2 — Timer 2 Overflow/Underflow — FF2002H OVRTM1 — Timer 1 Overflow/Underflow — FF2000H
2	Reserved; for compatibility with future devices, write zero to this bit.

INT_MASK1

INT_MASK1

Address: 0013H
Reset State: 00H

The interrupt mask 1 (INT_MASK1) register enables or disables (masks) individual interrupt requests. (The EI and DI instructions enable and disable servicing of all maskable interrupts.) INT_MASK1 can be read from or written to as a byte register. PUSHA saves this register on the stack and POPA restores it.

7							0
NMI	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1

Bit Number	Function
7:0	Setting a bit enables the corresponding interrupt. The standard interrupt vector locations are as follows: **Bit Mnemonic** — **Interrupt** — **Standard Vector** NMI — Nonmaskable Interrupt — FF203EH EXTINT3 — EXTINT3 pin — FF203CH EXTINT2 — EXTINT2 pin — FF203AH OVR2_3† — EPA Capture Channel 2 or 3 Overrun — FF2038H OVR0_1† — EPA Capture Channel 0 or 1 Overrun — FF2036H EPA3 — EPA Capture/Compare Channel 3 — FF2034H EPA2 — EPA Capture/Compare Channel 2 — FF2032H EPA1 — EPA Capture/Compare Channel 1 — FF2030H † An overrun on the EPA capture/compare channels can generate the multiplexed capture overrun interrupts. The EPA_MASK and EPA_PEND registers decode these multiplexed interrupts. Write to EPA_MASK to enable the interrupt sources; read EPA_PEND to determine which source caused the interrupt.

INT_PEND

INT_PEND

Address: 0009H
Reset State: 00H

When hardware detects a pending interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7 **0**

EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM2	OVRTM1

Bit Number	Function
7:3 1:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. The standard interrupt vector locations are as follows:
2	Reserved. This bit is undefined.

Bit Mnemonic	Interrupt	Standard Vector
EPA0	EPA Capture/Compare Channel 0	FF200EH
RI	SIO Receive	FF200CH
TI	SIO Transmit	FF200AH
EXTINT1	EXTINT1 pin	FF2008H
EXTINT0	EXTINT0 pin	FF2006H
OVRTM2	Timer 2 Overflow/Underflow	FF2002H
OVRTM1	Timer 1 Overflow/Underflow	FF2000H

INT_PEND1

INT_PEND1 Address: 0012H
Reset State: 00H

When hardware detects a pending interrupt, it sets the corresponding bit in the interrupt pending (INT_PEND or INT_PEND1) registers. When the vector is taken, the hardware clears the pending bit. Software can generate an interrupt by setting the corresponding interrupt pending bit.

7							0
NMI	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1

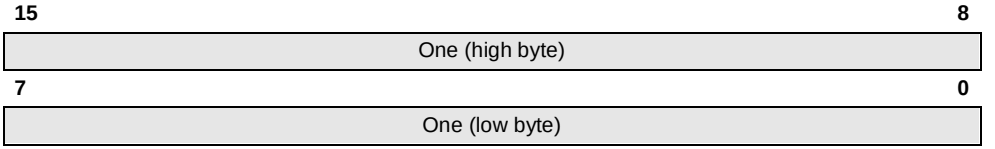
Bit Number	Function
7:0	Any set bit indicates that the corresponding interrupt is pending. The interrupt bit is cleared when processing transfers to the corresponding interrupt vector. The standard interrupt vector locations are as follows: **Bit Mnemonic** — **Interrupt** — **Standard Vector** NMI — Nonmaskable Interrupt — FF203EH EXTINT3 — EXTINT3 pin — FF203CH EXTINT2 — EXTINT2 pin — FF203AH OVR2_3† — EPA Capture Channel 2 or 3 Overrun — FF2038H OVR0_1† — EPA Capture Channel 0 or 1 Overrun — FF2036H EPA3 — EPA Capture/Compare Channel 3 — FF2034H EPA2 — EPA Capture/Compare Channel 2 — FF2032H EPA1 — EPA Capture/Compare Channel 1 — FF2030H † An overrun on the EPA capture/compare channels can generate the multiplexed capture overrun interrupts. The EPA_MASK and EPA_PEND registers decode these multiplexed interrupts. Write to EPA_MASK to enable the interrupt sources; read EPA_PEND to determine which source caused the interrupt.

ONES_REG

ONES_REG

Address: 02H
Reset State: FFFFH

The two-byte ones register (ONES_REG) is always equal to FFFFH. It is useful as a fixed source of all ones for comparison operations.



Bit Number	Function
15:0	One These bits are always equal to FFFFH.

Px_DIR

P*x*_DIR
***x* = 1–4**

Address: Table C-10
Reset State:

Each pin of port *x* can operate in any of the standard I/O modes of operation: complementary output, open-drain output, or high-impedance input. The port *x* I/O direction (P*x*_DIR) register determines the I/O direction for each port *x* pin. The register settings for an open-drain output or a high-impedance input are identical. An open-drain output configuration requires an external pull-up. A high-impedance input configuration requires that the corresponding bit in P*x*_REG be set.

	7							0
***x* = 1–3**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 4**	—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Port *x* Pin *y* Direction This bit selects the P*x.y* direction: 0 = complementary output (output only) 1 = input or open-drain output (input, output, or bidirectional Open-drain outputs require external pull-ups.

Table C-10. P*x*_DIR Addresses and Reset Values

Register	Address	Reset Value
P1_DIR	1FD2H	FFH
P2_DIR	1FD3H	FFH
P3_DIR	1FDAH	FFH
P4_DIR	1FDBH	FFH

Px_MODE

P*x*_MODE
***x* = 1–4**

Address: Table C-11
Reset State:

Each bit of the port *x* mode (P*x*_MODE) register controls whether the corresponding pin functions as a standard I/O port pin or as a special-function signal.

	7							0
***x* = 1–3**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 4**	—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Port *x* Pin *y* Mode This bit determines the mode of the corresponding port pin: 0 = standard I/O port pin 1 = special-function signal Table C-12 lists the special-function signals for each pin.

Table C-11. P*x*_MODE Addresses and Reset Values

Register	Address	Reset Value
P1_MODE	1FD0H	00H
P2_MODE	1FD1H	80H
P3_MODE	1FD8H	01H
P4_MODE	1FD9H	00H

Table C-12. Special-function Signals for Ports 1–4

Port 1	
Pin	**Special-function Signal**
P1.0	EPA0
P1.1	EPA1
P1.2	EPA2
P1.3	EPA3
P1.4	T1CLK
P1.5	T1DIR
P1.6	T2CLK
P1.7	T2DIR

Port 2	
Pin	**Special-function Signal**
P2.0	TXD
P2.1	RXD
P2.2	EXTINT0
P2.3	BREQ#
P2.4	EXTINT1
P2.5	HOLD#
P2.6	HLDA#
P2.7	CLKOUT

Port 3	
Pin	**Special-function Signal**
P3.0	CS0#
P3.1	CS1#
P3.2	CS2#
P3.3	CS3#
P3.4	CS4#
P3.5	CS5#
P3.6	EXTINT2
P3.7	EXTINT3

Port 4	
Pin	**Special-function Signal**
P4.0	PWM0
P4.1	PWM1
P4.2	PWM2
P4.3	—

Px_PIN

P*x*_PIN
***x* = 1–4**

Address: Table C-13
Reset State:

Each bit of the port *x* pin input (P*x*_PIN) register reflects the current state of the corresponding pin, regardless of the pin configuration.

	7							0
***x* = 1–3**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 4**	—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Port *x* Pin *y* Input Value This bit contains the current state of P*x.y.*

Table C-13. P*x*_PIN Addresses and Reset Values

Register	Address	Reset Value
P1_PIN	1FD6H	XXH
P2_PIN	1FD7H	XXH
P3_PIN	1FDEH	XXH
P4_PIN	1FDFH	XXH

Px_REG

P*x*_REG
***x* = 1–4**

Address: Table C-14
Reset State:

For an input, set the corresponding port x data ouput (P*x*_REG) register bit.

For an output, write the data to be driven out by each pin to the corresponding bit of P*x*_REG. When a pin is configured as standard I/O (P*x*_MODE.*y* = 0), the result of a CPU write to P*x*_REG is immediately visible on the pin. When a pin is configured as a special-function signal (P*x*_MODE.*y* = 1), the associated on-chip peripheral or off-chip component controls the pin. The CPU can still write to P*x*_REG, but the pin is unaffected until it is switched back to its standard I/O function.

This feature allows software to configure a pin as standard I/O (clear P*x*_MODE.*y*), initialize or overwrite the pin value, then configure the pin as a special-function signal (set P*x*_MODE.*y*). In this way, initialization, fault recovery, exception handling, etc., can be done without changing the operation of the associated peripheral.

	7							0
***x* = 1–3**	PIN7	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1	PIN0
	7							**0**
***x* = 4**	—	—	—	—	PIN3	PIN2	PIN1	PIN0

Bit Number	Bit Mnemonic	Function
7:0	PIN7:0	Port *x* Pin *y* Output To use P*x.y* for output, write the desired output data to this bit. To use P*x.y* for input, set this bit.

Table C-14. P*x*_REG Addresses and Reset Values

Register	Address	Reset Value
P1_REG	1FD4H	FFH
P2_REG	1FD5H	FFH
P3_REG	1FDCH	FFH
P4_REG	1FDDH	FFH

PSW

PSW no direct access

The processor status word (PSW) actually consists of two bytes. The high byte is the status word, which is described here; the low byte is the INT_MASK register. The status word contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of a user's program.

The status word portion of the PSW cannot be accessed directly. To access the status word, push the value onto the stack (PUSHF), then pop the value to a register (POP *test_reg*). The PUSHF and PUSHA instructions save the PSW in the system stack and then clear it; POPF and POPA restore it.

15							**8**
Z	N	V	VT	C	PSE	I	ST

7	**0**
See INT_MASK on page C-25	

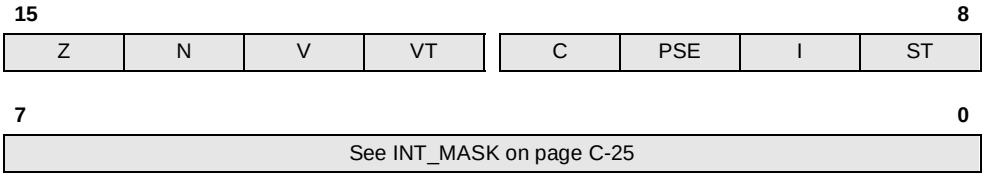
Bit Number	Bit Mnemonic	Function
7	Z	Zero Flag This flag is set to indicate that the result of an operation was zero. For multiple-precision calculations, the zero flag cannot be set by the instructions that use the carry bit from the previous calculation (e.g., ADDC, SUBC). However, these instructions can clear the zero flag. This ensures that the zero flag will reflect the result of the entire operation, not just the last calculation. For example, if the result of adding together the lower words of two double words is zero, the zero flag would be set. When the upper words are added together using the ADDC instruction, the flag remains set if the result is zero and is cleared if the result is not zero.
6	N	Negative Flag This flag is set to indicate that the result of an operation is negative. The flag is correct even if an overflow occurs. For all shift operations and the NORML instruction, the flag is set to equal the most-significant bit of the result, even if the shift count is zero.
5	V	Overflow Flag This flag is set to indicate that the result of an operation is too large to be represented correctly in the available space. For shift operations (SHL, SHLB, and SHLL), the flag is set if the most-significant bit of the operand changes during the shift. For divide operations, the quotient is stored in the low-order half of the destination operand and the remainder is stored in the high-order half. The overflow flag is set if the quotient is outside the range for the low-order half of the destination operand. (Chapter 4, "Programming Considerations," defines the operands and possible values for each. See the PSW flag descriptions in Appendix A for details.)

PSW

PSW (Continued) no direct access

The processor status word (PSW) actually consists of two bytes. The high byte is the status word, which is described here; the low byte is the INT_MASK register. The status word contains one bit (PSW.1) that globally enables or disables servicing of all maskable interrupts, one bit (PSW.2) that enables or disables the peripheral transaction server (PTS), and six Boolean flags that reflect the state of a user's program.

The status word portion of the PSW cannot be accessed directly. To access the status word, push the value onto the stack (PUSHF), then pop the value to a register (POP *test_reg*). The PUSHF and PUSHA instructions save the PSW in the system stack and then clear it; POPF and POPA restore it.



Bit Number	Bit Mnemonic	Function
4	VT	Overflow-trap Flag This flag is set when the overflow flag is set, but it is cleared only by the CLRVT, JVT, and JNVT instructions. This allows testing for a possible overflow at the end of a sequence of related arithmetic operations, which is generally more efficient than testing the overflow flag after each operation.
3	C	Carry Flag This flag is set to indicate an arithmetic carry or the last bit shifted out of an operand. It is cleared if a subtraction operation generates a borrow. Normally, the result is rounded up if the carry flag is set. The sticky bit flag allows a finer resolution in the rounding decision. (See the PSW flag descriptions in Appendix A for details.)
2	PSE	PTS Enable This bit globally enables or disables the peripheral transaction server (PTS). The EPTS instruction sets this bit; DPTS clears it. 1 = enable PTS 0 = disable PTS
1	I	Interrupt Disable (Global) This bit globally enables or disables the servicing of all *maskable interrupts*. The bits in INT_MASK and INT_MASK1 individually enable or disable the interrupts. The EI instruction sets this bit; DI clears it. 1 = enable interrupt servicing 0 = disable interrupt servicing
0	ST	Sticky Bit Flag This flag is set to indicate that, during a right shift, a "1" was shifted into the carry flag and then shifted out. It can be used with the carry flag to allow finer resolution in rounding decisions.

PTSSEL

PTSSEL

Address: 0004H
Reset State: 0000H

The PTS select (PTSSEL) register selects either a PTS microcode routine or a standard interrupt service routine for each interrupt request. Setting a bit selects a PTS microcode routine; clearing a bit selects a standard interrupt service routine. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit. The PTSSEL bit must be set manually to re-enable the PTS channel.

15							8
—	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1
7							**0**
EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM2	OVRTM1

Bit Number	Function
15, 2	Reserved; for compatibility with future devices, write zero to this bit.
14:3 1:0	Setting a bit causes the corresponding interrupt to be handled by a PTS microcode routine. The PTS interrupt vector locations are as follows:

Bit Mnemonic	Interrupt	PTS Vector
EXTINT3	EXTINT3 pin	FF205CH
EXTINT2	EXTINT2 pin	FF205AH
OVR2_3†	EPA Capture Channel 2 or 3 Overrun	FF2058H
OVR0_1†	EPA Capture Channel 0 or 1 Overrun	FF2056H
EPA3	EPA Capture/Compare Channel 3	FF2054H
EPA2	EPA Capture/Compare Channel 2	FF2052H
EPA1	EPA Capture/Compare Channel 1	FF2050H
EPA0	EPA Capture/Compare Channel 0	FF204EH
RI	SIO Receive	FF204CH
TI	SIO Transmit	FF204AH
EXTINT1	EXTINT1 pin	FF2048H
EXTINT0	EXTINT0 pin	FF2046H
OVRTM2	Timer 2 Overflow/ Underflow	FF2042H
OVRTM1	Timer 1 Overflow/ Underflow	FF2040H

† PTS service is not recommended because the PTS cannot determine the source of shared interrupts.

PTSSRV

PTSSRV

Address: 0006H
Reset State: 0000H

The PTS service (PTSSRV) register is used by the hardware to indicate that the final PTS interrupt has been serviced by the PTS routine. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt. When the end-of-PTS interrupt is called, hardware clears the PTSSRV bit. The PTSSEL bit must be set manually to re-enable the PTS channel.

15							8
—	EXTINT3	EXTINT2	OVR2_3	OVR0_1	EPA3	EPA2	EPA1
7							**0**
EPA0	RI	TI	EXTINT1	EXTINT0	—	OVRTM1	OVRTM2

Bit Number	Function
15, 2	Reserved. These bits are undefined.
14:3 1:0	A bit is set by hardware to request an end-of-PTS interrupt for the corresponding interrupt through its standard interrupt vector. The standard interrupt vector locations are as follows.

Bit Mnemonic	Interrupt	Standard Vector
EXTINT3	EXTINT3 Pin	FF203CH
EXTINT2	EXTINT2 Pin	FF203AH
OVR2_3†	EPA Capture Channel 2 or 3 Overrun	FF2038H
OVR0_1†	EPA Capture Channel 0 or 1 Overrun	FF2036H
EPA3	EPA Capture/Compare Channel 3	FF2034H
EPA2	EPA Capture/Compare Channel 2	FF2032H
EPA1	EPA Capture/Compare Channel 1	FF2030H
EPA0	EPA Capture/Compare Channel 0	FF200EH
RI	SIO Receive	FF200CH
TI	SIO Transmit	FF200AH
EXTINT1	EXTINT1 pin	FF2008H
EXTINT0	EXTINT0 pin	FF2006H
OVRTM2	Timer 2 Overflow/Underflow	FF2002H
OVRTM1	Timer 1 Overflow/Underflow	FF2000H

† PTS service is not recommended for multiplexed interrupts. This bit is cleared when both corresponding interrupt pending bits are cleared in EPA_PEND.

PWMx_CONTROL

PWM*x*_CONTROL
***x* = 0–2**

Address: Table C-15
Reset State:

The PWM control (PWM*x*_CONTROL) register determines the duty cycle of the PWM *x* channel. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).

7							0
PWM Duty Cycle							

Bit Number	Function
7:0	PWM Duty Cycle This register controls the PWM duty cycle. A zero loaded into this register causes the PWM to output a low continuously (0% duty cycle). An FFH in this register causes the PWM to have its maximum duty cycle (99.6% duty cycle).

Table C-15. PWM*x*_CONTROL Addresses and Reset Values

Register	Address	Reset Value
PWM0_CONTROL	1FB0H	00H
PWM1_CONTROL	1FB2H	00H
PWM2_CONTROL	1FB4H	00H

SBUF_RX

SBUF_RX

Address: 1FB8H
Reset State: 00H

The serial port receive buffer (SBUF_RX) register contains data received from the serial port. The serial port receiver is buffered and can begin receiving a second data byte before the first byte is read. Data is held in the receive shift register until the last data bit is received, then the data byte is loaded into SBUF_RX. If data in the shift register is loaded into SBUF_RX before the previous byte is read, the overflow error bit is set (SP_STATUS.2). The data in SBUF_RX will always be the last byte received, never a combination of the last two bytes.

7							0
Data Received							

Bit Number	Function
7:0	Data Received This register contains the last byte of data received from the serial port.

SBUF_TX

SBUF_TX Address: 1FBAH
Reset State: 00H

The serial port transmit buffer (SBUF_TX) register contains data that is ready for transmission. In modes 1, 2, and 3, writing to SBUF_TX starts a transmission. In mode 0, writing to SBUF_TX starts a transmission only if the receiver is disabled (SP_CON.3=0).

7							0
Data to Transmit							

Bit Number	Function
7:0	Data to Transmit This register contains a byte of data to be transmitted by the serial port.

SP

SP Address: 18H
Reset State: XXXXH

The system's stack pointer (SP) can point anywhere in an internal or external memory page; it must be word aligned and must always be initialized before use. The stack pointer is decremented before a PUSH and incremented after a POP, so the stack pointer should be initialized to two bytes (in 64-Kbyte mode) or four bytes (in 1-Mbyte mode) above the highest stack location. If stack operations are not being performed, locations 18H and 19H may be used as standard registers.

15	8
Stack Pointer (high byte)	

7	0
Stack Pointer (low byte)	

Bit Number	Function
15:0	Stack Pointer This register makes up the system's stack pointer.

SP_BAUD

SP_BAUD

Address: 1FBCH
Reset State: 0000H

The serial port baud rate (SP_BAUD) register selects the serial port baud rate and clock source. The most-significant bit selects the clock source. The lower 15 bits represent BAUD_VALUE, an unsigned integer that determines the baud rate.

The maximum BAUD_VALUE is 32,767 (7FFFH). In asynchronous modes 1, 2, and 3, the minimum BAUD_VALUE is 0000H when using the internal clock source (f) and 0001H when using T1CLK. In synchronous mode 0, the minimum BAUD_VALUE is 0001H for transmissions and 0002H for receptions.

15							**8**
CLKSRC	BV14	BV13	BV12	BV11	BV10	BV9	BV8
7							**0**
BV7	BV6	BV5	BV4	BV3	BV2	BV1	BV0

Bit Number	Bit Mnemonic	Function
15	CLKSRC	Serial Port Clock Source This bit determines whether the serial port is clocked from an internal or an external source. 0 = signal on the T1CLK pin (external source) 1 = internal operating frequency (f)
14:0	BV14:0	Baud Rate These bits constitute the BAUD_VALUE. Use the following equations to determine the BAUD_VALUE for a given baud rate. Synchronous mode 0:† $BAUD\_VALUE = \frac{f}{Baud\ Rate \times 2} - 1$ or $\frac{T1CLK}{Baud\ Rate}$ Asynchronous modes 1, 2, and 3: $BAUD\_VALUE = \frac{f}{Baud\ Rate \times 16} - 1$ or $\frac{T1CLK}{Baud\ Rate \times 8}$ † For mode 0 receptions, the BAUD_VALUE must be 0002H or greater. Otherwise, the resulting data in the receive shift register will be incorrect.

Table C-16. SP_BAUD Values When Using the Internal Clock at 25 MHz

Baud Rate	SP_BAUD Register Value (Note 1)		% Error	
	Mode 0	Mode 1, 2, 3	Mode 0	Mode 1, 2, 3
9600	8515H	80A2H	0	0.15
4800	8A2BH	8144H	0	0.16
2400	9457H	828AH	0	0
1200	A8AFH	8515H	0	0
300	(Note 2)	9457H	(Note 2)	0

NOTES:

1. Bit 15 is always set when the internal peripheral clock is selected as the clock source for the baud-rate generator.
2. For mode 0 operation at 25 MHz, the minimum baud rate is 381.47 (BAUD_VALUE = 7FFFH). For mode 0 operation at 300 baud, the maximum internal clock frequency is 19.6608 MHz (BAUD_VALUE = 7FFFH).

SP_CON

SP_CON Address: 1FBBH
Reset State: 00H

The serial port control (SP_CON) register selects the communications mode and enables or disables the receiver, parity checking, and nine-bit data transmission. For the 80C196NU, it also enables or disables the divide-by-two prescaler.

	7							0
8XC196NP	—	—	PAR	TB8	REN	PEN	M1	M0
	7							**0**
80C196NU	—	PRS	PAR	TB8	REN	PEN	M1	M0

Bit Number	Bit Mnemonic	Function
7	—	Reserved; for compatibility with future devices, write zero to this bit.
6†	PRS	Prescale This bit enables the divide-by-two prescaler. 0 = disable the prescaler 1 = enable the prescaler
5	PAR	Parity Selection Bit Selects even or odd parity. 0 = even parity 1 = odd parity
4	TB8	Transmit Ninth Data Bit This is the ninth data bit that will be transmitted in mode 2 or 3. This bit is cleared after each transmission, so it must be set before SBUF_TX is written. When SP_CON.2 is set, this bit takes on the even parity value.
3	REN	Receive Enable Setting this bit enables the receiver function of the RXD pin. When this bit is set, a high-to-low transition on the pin starts a reception in mode 1, 2, or 3. In mode 0, this bit must be clear for transmission to begin and must be set for reception to begin. Clearing this bit stops a reception in progress and inhibits further receptions.
2	PEN	Parity Enable In modes 1 and 3, setting this bit enables the parity function. This bit must be cleared if mode 2 is used. When this bit is set, TB8 takes the parity value on transmissions. With parity enabled, SP_STATUS.7 becomes the receive parity error bit.
1:0	M1:0	Mode Selection These bits select the communications mode. **M1 M0** 0 0 mode 0 0 1 mode 1 1 0 mode 2 1 1 mode 3

† This bit is reserved on the 8XC196NP. For compatibility with future devices, write zero to this bit.

SP_STATUS

SP_STATUS

Address: 1FB9H
Reset State: 0BH

The serial port status (SP_STATUS) register contains bits that indicate the status of the serial port.

7							0
RPE/RB8	RI	TI	FE	TXE	OE	—	—

Bit Number	Bit Mnemonic	Function
7	RPE/RB8	Received Parity Error/Received Bit 8 RPE is set if parity is disabled (SP_CON.2 = 0) and the ninth data bit received is high. RB8 is set if parity is enabled (SP_CON.2 = 1) and a parity error occurred. Reading SP_STATUS clears this bit.
6	RI	Receive Interrupt This bit is set when the last data bit is sampled. Reading SP_STATUS clears this bit. This bit need **not** be clear for the serial port to receive data.
5	TI	Transmit Interrupt This bit is set at the beginning of the stop bit transmission. Reading SP_STATUS clears this bit.
4	FE	Framing Error This bit is set if a stop bit is not found within the appropriate period of time. Reading SP_STATUS clears this bit.
3	TXE	SBUF_TX Empty This bit is set if the transmit buffer is empty and ready to accept up to two bytes. It is cleared when a byte is written to SBUF_TX.
2	OE	Overrun Error This bit is set if data in the receive shift register is loaded into SBUF_RX before the previous bit is read. Reading SP_STATUS clears this bit.
1:0	—	Reserved. These bits are undefined.

T1CONTROL

T1CONTROL Address: 1F90H
Reset State: 00H

The timer 1 control (T1CONTROL) register determines the clock source, counting direction, and count rate for timer 1.

7 **0**

CE	UD	M2	M1	M0	P2	P1	P0

Bit Number	Bit Mnemonic	Function
7	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disables timer 1 = enables timer
6	UD	Up/Down This bit determines the timer counting direction, in selected modes (see mode bits, M2:0). 0 = count down 1 = count up
5:3	M2:0	EPA Clock Direction Mode Bits These bits determine the timer clocking source and direction control source. **M2 M1 M0 Clock Source Direction Source** 0 0 0 f/4 UD bit (T1CONTROL.6) X 0 1 T1CLK pin† UD bit (T1CONTROL.6) 0 1 0 f/4 T1DIR pin 0 1 1 T1CLK pin† T1DIR pin 1 1 1 quadrature clocking using T1CLK and T1DIR † If an external clock is selected, the timer counts on both the rising **and** falling edges of the clock.
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. **P2 P1 P0 Prescaler Divisor Resolution†** 0 0 0 divide by 1 (disabled) 160 ns 0 0 1 divide by 2 320 ns 0 1 0 divide by 4 640 ns 0 1 1 divide by 8 1.28 µs 1 0 0 divide by 16 2.56 µs 1 0 1 divide by 32 5.12 µs 1 1 0 divide by 64 10.24 µs 1 1 1 divide by 128 (NU only) 20.48 µs † At f = 25 MHz. Use the formula on page 10-6 to calculate the resolution at other frequencies.

T2CONTROL

T2CONTROL

Address: 1F94H
Reset State: 00H

The timer 2 control (T2CONTROL) register determines the clock source, counting direction, and count rate for timer 2.

7 **0**

CE	UD	M2	M1	M0	P2	P1	P0

Bit Number	Bit Mnemonic	Function
7	CE	Counter Enable This bit enables or disables the timer. From reset, the timers are disabled and not free running. 0 = disables timer 1 = enables timer
6	UD	Up/Down This bit determines the timer counting direction, in selected modes (see mode bits, M2:0). 0 = count down 1 = count up
5:3	M2:0	EPA Clock Direction Mode Bits. These bits determine the timer clocking source and direction source **M2 M1 M0 Clock Source Direction Source** 0 0 0 f/4 UD bit (T2CONTROL.6) X 0 1 T2CLK pin† UD bit (T2CONTROL.6) 0 1 0 f/4 T2DIR pin 0 1 1 T2CLK pin† T2DIR pin 1 0 0 timer 1 overflow UD bit (T2CONTROL.6) 1 1 0 timer 1 same as timer 1 1 1 1 quadrature clocking using T2CLK and T2DIR † If an external clock is selected, the timer counts on both the rising **and** falling edges of the clock.
2:0	P2:0	EPA Clock Prescaler Bits These bits determine the clock prescaler value. **P2 P1 P0 Prescaler Resolution†** 0 0 0 divide by 1 (disabled) 160 ns 0 0 1 divide by 2 320 ns 0 1 0 divide by 4 640 ns 0 1 1 divide by 8 1.28 µs 1 0 0 divide by 16 2.56 µs 1 0 1 divide by 32 5.12 µs 1 1 0 divide by 64 10.24 µs 1 1 1 divide by 128 (NU only) 20.48 µs † At f = 25 MHz. Use the formula on page 10-6 to calculate the resolution at other frequencies.

TIMERx

TIMER*x*
***x* = 1–2**

Address: Table C-17
Reset State:

Th*is* register contains the value of timer *x*. This register can be written, allowing timer *x* to be initialized to a value other than zero.

15	8
Timer Value (high byte)	
7	**0**
Timer Value (low byte)	

Bit Number	Function
15:0	Timer Read the current timer *x* value from this register or write a new timer *x* value to this register.

Table C-17. TIMER*x* Addresses and Reset Values

Register	Address	Reset Value
TIMER1	1F92H	0000H
TIMER2	1F96H	0000H

WSR

WSR Address: 0014H
Reset State: 00H

The window selection register (WSR) has two functions. One bit enables and disables the bus-hold protocol. The remaining bits select windows. Windows map sections of RAM into the top of the lower register file, in 32-, 64-, or 128-byte increments. PUSHA saves this register on the stack and POPA restores it.

7							0
HLDEN	W6	W5	W4	W3	W2	W1	W0

Bit Number	Bit Mnemonic	Function
7	HLDEN	HOLD#, HLDA# Protocol Enable This bit enables and disables the bus-hold protocol (see Chapter 13, “Interfacing with External Memory”). It has no effect on windowing. 1 = enable 0 = disable
6:0	W6:0	Window Selection These bits specify the window size and window number. See Table 5-8 on page 5-15 or Table 5-9 on page 5-15.

Table C-18. WSR Settings and Direct Addresses for Windowable SFRs

Register Mnemonic	Memory Location	32-byte Windows (00E0–00FFH)		64-byte Windows (00C0–00FFH)		128-byte Windows (0080–00FFH)	
		WSR	Direct Address	WSR	Direct Address	WSR	Direct Address
ADDRCOM0†	1F40H	7AH	00E0H	3DH	00C0H	1EH	00C0H
ADDRCOM1†	1F48H	7AH	00E8H	3DH	00C8H	1EH	00C8H
ADDRCOM2†	1F50H	7AH	00F0H	3DH	00D0H	1EH	00D0H
ADDRCOM3†	1F58H	7AH	00F8H	3DH	00D8H	1EH	00D8H
ADDRCOM4†	1F60H	7BH	00E0H	3DH	00E0H	1EH	00E0H
ADDRCOM5†	1F68H	7BH	00E8H	3DH	00E8H	1EH	00E8H
ADDRMSK0†	1F42H	7AH	00E2H	3DH	00C2H	1EH	00C2H
ADDRMSK1†	1F4AH	7AH	00EAH	3DH	00CAH	1EH	00CAH
ADDRMSK2†	1F52H	7AH	00F2H	3DH	00D2H	1EH	00D2H
ADDRMSK3†	1F5AH	7AH	00FAH	3DH	00DAH	1EH	00DAH
ADDRMSK4†	1F62H	7BH	00E2H	3DH	00E2H	1EH	00E2H
ADDRMSK5†	1F6AH	7BH	00EAH	3DH	00EAH	1EH	00EAH
BUSCON0	1F44H	7AH	00E4H	3DH	00C4H	1EH	00C4H

† Must be addressed as a word.

WSR

Table C-18. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

Register Mnemonic	Memory Location	32-byte Windows (00E0–00FFH)		64-byte Windows (00C0–00FFH)		128-byte Windows (0080–00FFH)	
		WSR	Direct Address	WSR	Direct Address	WSR	Direct Address
BUSCON1	1F4CH	7AH	00ECH	3DH	00CCH	1EH	00CCH
BUSCON2	1F54H	7AH	00F4H	3DH	00D4H	1EH	00D4H
BUSCON3	1F5CH	7AH	00FCH	3DH	00DCH	1EH	00DCH
BUSCON4	1F64H	7BH	00E4H	3DH	00E4H	1EH	00E4H
BUSCON5	1F6CH	7BH	00ECH	3DH	00ECH	1EH	00ECH
CON_REG0	1FB6H	7DH	00F6H	3EH	00F6H	1FH	00B6H
EP_DIR	1FE3H	7FH	00E3H	3FH	00E3H	1FH	00E3H
EP_MODE	1FE1H	7FH	00E1H	3FH	00E1H	1FH	00E1H
EP_PIN	1FE7H	7FH	00E7H	3FH	00E7H	1FH	00E7H
EP_REG	1FE5H	7FH	00E5H	3FH	00E5H	1FH	00E5H
EPA_MASK†	1F9CH	7CH	00FCH	3EH	00DCH	1FH	009CH
EPA_PEND	1F9EH	7CH	00FEH	3EH	00DEH	1FH	009EH
EPA0_CON	1F80H	7CH	00E0H	3EH	00C0H	1FH	0080H
EPA1_CON†	1F84H	7CH	00E4H	3EH	00C4H	1FH	0084H
EPA2_CON	1F88H	7CH	00E8H	3EH	00C8H	1FH	0088H
EPA3_CON†	1F8CH	7CH	00ECH	3EH	00CCH	1FH	008CH
EPA0_TIME†	1F82H	7CH	00E2H	3EH	00C2H	1FH	0082H
EPA1_TIME†	1F86H	7CH	00E6H	3EH	00C6H	1FH	0086H
EPA2_TIME†	1F8AH	7CH	00EAH	3EH	00CAH	1FH	008AH
EPA3_TIME†	1F8EH	7CH	00EEH	3EH	00CEH	1FH	008EH
P1_DIR	1FD2H	7EH	00F2H	3FH	00D2H	1FH	00D2H
P1_MODE	1FD0H	7EH	00F0H	3FH	00D0H	1FH	00D0H
P1_PIN	1FD6H	7EH	00F6H	3FH	00D6H	1FH	00D6H
P1_REG	1FD4H	7EH	00F4H	3FH	00D4H	1FH	00D4H
P2_DIR	1FD3H	7EH	00F3H	3FH	00D3H	1FH	00D3H
P2_MODE	1FD1H	7EH	00F1H	3FH	00D1H	1FH	00D1H
P2_PIN	1FD7H	7EH	00F7H	3FH	00D7H	1FH	00D7H
P2_REG	1FD5H	7EH	00F5H	3FH	00D5H	1FH	00D5H
P3_DIR	1FDAH	7EH	00FAH	3FH	00DAH	1FH	00DAH
P3_MODE	1FD8H	7EH	00F8H	3FH	00D8H	1FH	00D8H
P3_PIN	1FDEH	7EH	00FEH	3FH	00DEH	1FH	00DEH
P3_REG	1FDCH	7EH	00FCH	3FH	00DCH	1FH	00DCH

† Must be addressed as a word.

Table C-18. WSR Settings and Direct Addresses for Windowable SFRs (Continued)

Register Mnemonic	Memory Location	32-byte Windows (00E0–00FFH)		64-byte Windows (00C0–00FFH)		128-byte Windows (0080–00FFH)	
		WSR	Direct Address	WSR	Direct Address	WSR	Direct Address
P4_DIR	1FDBH	7EH	00FBH	3FH	00DBH	1FH	00DBH
P4_MODE	1FD9H	7EH	00F9H	3FH	00D9H	1FH	00D9H
P4_PIN	1FDFH	7EH	00FFH	3FH	00DFH	1FH	00DFH
P4_REG	1FDDH	7EH	00FDH	3FH	00DDH	1FH	00DDH
PWM0_CONTROL	1FB0H	7DH	00F0H	3EH	00F0H	1FH	00B0H
PWM1_CONTROL	1FB2H	7DH	00F2H	3EH	00F2H	1FH	00B2H
PWM2_CONTROL	1FB4H	7DH	00F4H	3EH	00F4H	1FH	00B4H
SBUF_RX	1FB8H	7DH	00F8H	3EH	00F8H	1FH	00B8H
SBUF_TX	1FBAH	7DH	00FAH	3EH	00FAH	1FH	00BAH
SP_BAUD	1FBCH	7DH	00FCH	3EH	00FCH	1FH	00BCH
SP_CON	1FBBH	7DH	00FBH	3EH	00FBH	1FH	00BBH
SP_STATUS	1FB9H	7DH	00F9H	3EH	00F9H	1FH	00B9H
T1CONTROL	1F90H	7CH	00F0H	3EH	00D0H	1FH	0090H
T2CONTROL	1F94H	7CH	00F4H	3EH	00D4H	1FH	0094H
TIMER1†	1F92H	7CH	00F2H	3EH	00D2H	1FH	0092H
TIMER2†	1F96H	7CH	00F6H	3EH	00D6H	1FH	0096H

† Must be addressed as a word.

WSR1

WSR1
(80C196NU)

Address: 0015H
Reset State: 00H

Window selection 1 (WSR1) register selects a 32- or 64-byte segment of the upper register file or peripheral SFRs to be windowed into the middle of the lower register file, below any window selected by the WSR.

	7							0
80C196NU	—	W6	W5	W4	W3	W2	W1	W0

Bit Number	Bit Mnemonic	Function
7	—	Reserved; always write as zero.
6:0	W6:0	Window Selection These bits specify the window size and window number. See Table 5-8 on page 5-15 or Table 5-9 on page 5-15.

Table C-19. WSR1 Settings and Direct Addresses for Windowable SFRs

Register Mnemonic	Memory Location	32-byte Windows (0060–007FH)		64-byte Windows (0040–007FH)	
		WSR1	Direct Address	WSR1	Direct Address
ADDRCOM0†	1F40H	7AH	0060H	3DH	0040H
ADDRCOM1†	1F48H	7AH	0068H	3DH	0048H
ADDRCOM2†	1F50H	7AH	0070H	3DH	0050H
ADDRCOM3†	1F58H	7AH	0078H	3DH	0058H
ADDRCOM4†	1F60H	7BH	0060H	3DH	0060H
ADDRCOM5†	1F68H	7BH	0068H	3DH	0068H
ADDRMSK0†	1F42H	7AH	0062H	3DH	0042H
ADDRMSK1†	1F4AH	7AH	006AH	3DH	004AH
ADDRMSK2†	1F52H	7AH	0072H	3DH	0052H
ADDRMSK3†	1F5AH	7AH	007AH	3DH	005AH
ADDRMSK4†	1F62H	7BH	0062H	3DH	0062H
ADDRMSK5†	1F6AH	7BH	006AH	3DH	006AH
BUSCON0	1F44H	7AH	0064H	3DH	0044H
BUSCON1	1F4CH	7AH	006CH	3DH	004CH
BUSCON2	1F54H	7AH	0074H	3DH	0054H
BUSCON3	1F5CH	7AH	007CH	3DH	005CH
BUSCON4	1F64H	7BH	0064H	3DH	0064H

† Must be addressed as a word.

Table C-19. WSR1 Settings and Direct Addresses for Windowable SFRs (Continued)

Register Mnemonic	Memory Location	32-byte Windows (0060–007FH)		64-byte Windows (0040–007FH)	
		WSR1	Direct Address	WSR1	Direct Address
BUSCON5	1F6CH	7BH	006CH	3DH	006CH
CON_REG0	1FB6H	7DH	0076H	3EH	0076H
EP_DIR	1FE3H	7FH	0063H	3FH	0063H
EP_MODE	1FE1H	7FH	0061H	3FH	0061H
EP_PIN	1FE7H	7FH	0067H	3FH	0067H
EP_REG	1FE5H	7FH	0065H	3FH	0065H
EPA_MASK†	1F9CH	7CH	007CH	3EH	005CH
EPA_PEND	1F9EH	7CH	007EH	3EH	005EH
EPA0_CON	1F80H	7CH	0060H	3EH	0040H
EPA0_TIME†	1F82H	7CH	0062H	3EH	0042H
EPA1_CON†	1F84H	7CH	0064H	3EH	0044H
EPA1_TIME†	1F86H	7CH	0066H	3EH	0046H
EPA2_CON	1F88H	7CH	0068H	3EH	0048H
EPA2_TIME†	1F8AH	7CH	006AH	3EH	004AH
EPA3_CON†	1F8CH	7CH	006CH	3EH	004CH
EPA3_TIME†	1F8EH	7CH	006EH	3EH	004EH
P1_DIR	1FD2H	7EH	0072H	3FH	0052H
P1_MODE	1FD0H	7EH	0070H	3FH	0050H
P1_PIN	1FD6H	7EH	0076H	3FH	0056H
P1_REG	1FD4H	7EH	0074H	3FH	0054H
P2_DIR	1FD3H	7EH	0073H	3FH	0053H
P2_MODE	1FD1H	7EH	0071H	3FH	0051H
P2_PIN	1FD7H	7EH	0077H	3FH	0057H
P2_REG	1FD5H	7EH	0075H	3FH	0055H
P3_DIR	1FDAH	7EH	007AH	3FH	005AH
P3_MODE	1FD8H	7EH	0078H	3FH	0058H
P3_PIN	1FDEH	7EH	007EH	3FH	005EH
P3_REG	1FDCH	7EH	007CH	3FH	005CH
P4_DIR	1FDBH	7EH	007BH	3FH	005BH
P4_MODE	1FD9H	7EH	0079H	3FH	0059H
P4_PIN	1FDFH	7EH	007FH	3FH	005FH
P4_REG	1FDDH	7EH	007DH	3FH	005DH

† Must be addressed as a word.

WSR1

Table C-19. WSR1 Settings and Direct Addresses for Windowable SFRs (Continued)

Register Mnemonic	Memory Location	32-byte Windows (0060–007FH)		64-byte Windows (0040–007FH)	
		WSR1	Direct Address	WSR1	Direct Address
PWM0_CONTROL	1FB0H	7DH	0070H	3EH	0070H
PWM1_CONTROL	1FB2H	7DH	0072H	3EH	0072H
PWM2_CONTROL	1FB4H	7DH	0074H	3EH	0074H
SBUF_RX	1FB8H	7DH	0078H	3EH	0078H
SBUF_TX	1FBAH	7DH	007AH	3EH	007AH
SP_BAUD	1FBCH	7DH	007CH	3EH	007CH
SP_CON	1FBBH	7DH	007BH	3EH	007BH
SP_STATUS	1FB9H	7DH	0079H	3EH	0079H
T1CONTROL	1F90H	7CH	0070H	3EH	0050H
T2CONTROL	1F94H	7CH	0074H	3EH	0054H
TIMER1†	1F92H	7CH	0072H	3EH	0052H
TIMER2†	1F96H	7CH	0076H	3EH	0056H

† Must be addressed as a word.

ZERO_REG

ZERO_REG

Address: 00H
Reset State: 0000H

The two-byte zero register (ZERO_REG) is always equal to zero. It is useful as a fixed source of the constant zero for comparisons and calculations.

15	8
Zero (high byte)	

7	0
Zero (low byte)	

Bit Number	Function
15:0	Zero This register is always equal to zero.

Glossary

GLOSSARY

This glossary defines acronyms, abbreviations, and terms that have special meaning in this manual. (Chapter 1 discusses notational conventions and general terminology.)

1-Mbyte mode — The addressing mode that allows code to reside anywhere in the 1-Mbyte addressing space.

64-Kbyte mode — The addressing mode that allows code to reside only in page FFH.

accumulator — A register or storage location that forms the result of an arithmetic or logical operation.

The 80C196NU has enhanced multiplication instructions that use a new 32-bit accumulator for multiply-accumulate operations.

ALU — Arithmetic-logic unit. The part of the *RALU* that processes arithmetic and logical operations.

assert — The act of making a signal active (enabled). The polarity (high or low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low; to assert ALE is to drive it high.

bit — A binary digit.

BIT — A single-bit operand that can take on the Boolean values, “true” and “false.”

byte — Any 8-bit unit of data.

BYTE — An unsigned, 8-bit variable with values from 0 through 2^8–1.

CCBs — Chip configuration bytes. The chip configuration registers (*CCRs*) are loaded with the contents of the CCBs after a device reset.

CCRs — Chip configuration registers. Registers that define the environment in which the device will be operating. The chip configuration registers are loaded with the contents of the *CCBs* after a device reset.

chip-select unit — The integrated module that selects an external memory device during an external bus cycle.

clear — The "0" value of a bit or the act of giving it a "0" value. See also *set*.

deassert — The act of making a signal inactive (disabled). The polarity (high or low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To deassert RD# is to drive it high; to deassert ALE is to drive it low.

demultiplexed bus — The configuration in which the device uses separate lines for address and data (address on A19:0; data on AD15:0 for a 16-bit bus or AD7:0 for an 8-bit bus). See also *multiplexed bus*.

doping — The process of introducing a periodic table Group III or Group V element into a Group IV element (e.g., silicon). A Group III impurity (e.g., indium or gallium) results in a *p-type* material. A Group V impurity (e.g., arsenic or antimony) results in an *n-type* material.

double-word — Any 32-bit unit of data.

DOUBLE-WORD — An unsigned, 32-bit variable with values from 0 through $2^{32}-1$.

EDAR — Extended data address register used by the *EPORT*.

EPA — Event processor array. An integrated peripheral that provides high-speed input/output capability.

EPC — Extended program counter used by the *EPORT*.

EPORT — Extended addressing port. The port that provides the additional address lines to support extended addressing.

ESD — Electrostatic discharge.

external address — A 20-bit address is presented on the device pins. The address decoded by an external device depends on how many of these address lines the external system uses. See also *internal address*.

far constants — Constants that can be accessed only with extended instructions. See also *near constants*.

far data Data that can be accessed only with extended instructions. See also *near data*.

FET Field-effect transistor.

f Lowercase "f" represents the frequency of the internal clock. For the 8XC196NP, f is always equal to F_{XTAL1} (the input frequency on XTAL1). For the 80C196NU, which employs a phase-locked loop with clock multiplier circuitry, f is equal to either F_{XTAL1}, $2F_{XTAL1}$, or $4F_{XTAL1}$. The multiplier depends on the clock mode, which is controlled by the PLLEN1 and PLLEN2 input pins. (Figure 2-4 on page 2-8 illustrates the clock circuitry of the 80C196NU.)

fractional mode A mode of the *multiply-accumulate* function in which the multiplier result is shifted left one bit before being written to the *accumulator*. This left shift eliminates the extra sign bit when both operands are signed, leaving a correctly signed result.

hold latency The time it takes the microcontroller to assert HLDA# after an external device asserts HOLD#.

input leakage Current leakage from an input pin to power or ground.

integer Any member of the set consisting of the positive and negative whole numbers and zero.

INTEGER A 16-bit, signed variable with values from -2^{15} through $+2^{15}-1$.

internal address The 24-bit address that the microcontroller generates. See also *external address*.

interrupt controller The module responsible for handling interrupts that are to be serviced by *interrupt service routines* that you provide. Also called the *programmable interrupt controller (PIC)*.

interrupt latency The total delay between the time that an interrupt is generated (not acknowledged) and the time that the device begins executing the *interrupt service routine* or *PTS routine*.

interrupt service routine A software routine that you provide to service a standard interrupt. See also *PTS routine*.

interrupt vector A location in *special-purpose memory* that holds the starting address of an *interrupt service routine*.

ISR — See *interrupt service routine*.

LONG-INTEGER — A 32-bit, signed variable with values from -2^{31} through $+2^{31}-1$.

LSB — Least-significant bit of a byte or least-significant byte of a word.

MAC — See *multiply-accumulate*.

maskable interrupts — All interrupts except unimplemented opcode, software trap, and NMI. Maskable interrupts can be disabled (masked) by the individual mask bits in the interrupt mask registers, and their servicing can be disabled by the global interrupt enable bit. Each *maskable interrupt* can be assigned to the *PTS* for processing.

MSB — Most-significant bit of a *byte* or most-significant byte of a *word*.

multiplexed bus — The configuration in which the device uses both A19:0 and AD15:0 for address and also uses AD15:0 for data. See also *demultiplexed bus*.

multiply-accumulate — An operation performed by the 8XC196NU's enhanced multiplication instructions. The result of the operation is stored in a dedicated, 32-bit accumulator.

***n*-channel FET** — A field-effect transistor with an *n*-type conducting path (channel).

***n*-type material** — Semiconductor material with introduced impurities (*doping*) causing it to have an excess of negatively charged carriers.

near constants — Constants that can be accessed with nonextended instructions. Constants in page 00H are near constants (EP_REG = 00H is assumed). See also *far constants*.

near data — Data that can be accessed with nonextended instructions. Data in page 00H is near data (EP_REG = 00H is assumed). See also *far data*.

nonmaskable interrupts — Interrupts that cannot be masked (disabled) and cannot be assigned to the PTS for processing. The nonmaskable interrupts are unimplemented opcode, software trap, and NMI.

nonvolatile memory — Read-only memory that retains its contents when power is removed. Many MCS® 96 microcontrollers are available with either masked ROM, *EPROM*, or *OTPROM*. Consult the *Automotive Products* or *Embedded Microcontrollers* databook to determine which type of memory is available for a specific device.

***npn* transistor** — A transistor consisting of one part *p*-type material and two parts *n*-type material.

***p*-channel FET** — A field-effect transistor with a *p*-type conducting path.

***p*-type material** — Semiconductor material with introduced impurities (*doping*) causing it to have an excess of positively charged carriers.

PC — Program counter.

phase-locked loop — A component of the clock generation circuitry. The phase-locked loop (PLL) and the two input pins (PLLEN1 and PLLEN2) combine to enable the device to attain its maximum operating frequency with an external clock whose frequency is either equal to, one-half, or one-fourth that maximum frequency or with an external oscillator whose frequency is either one-half or one-fourth that maximum frequency.

PIC — Programmable interrupt controller. The module responsible for handling interrupts that are to be serviced by *interrupt service routines* that you provide. Also called simply the *interrupt controller.*

PLL — See *phase-locked loop*.

prioritized interrupt — Any *maskable interrupt* or nonmaskable NMI. Two of the *nonmaskable interrupts* (unimplemented opcode and software trap) are not prioritized; they vector directly to the *interrupt service routine* when executed.

program memory — A partition of memory where instructions can be stored for fetching and execution.

protected instruction — An instruction that prevents an interrupt from being acknowledged until after the next instruction executes. The protected instructions are DI, EI, DPTS, EPTS, POPA, POPF, PUSHA, and PUSHF.

PSW Processor status word. The high byte of the PSW is the status byte, which contains one bit that globally enables or disables servicing of all maskable interrupts, one bit that enables or disables the *PTS*, and six Boolean flags that reflect the state of the current program. The low byte of the PSW is the INT_MASK register. A push or pop instruction saves or restores both bytes (PSW + INT_MASK).

PTS Peripheral transaction server. The microcoded hardware interrupt processor.

PTSCB See *PTS control block*.

PTS control block A block of data required for each *PTS interrupt*. The microcode executes the proper *PTS routine* based on the contents of the PTS control block.

PTS cycle The microcoded response to a **single** PTS interrupt request.

PTS interrupt Any *maskable interrupt* that is assigned to the *PTS* for interrupt processing.

PTS mode A microcoded response that enables the *PTS* to complete a specific task quickly. These tasks include transferring a single byte or word, transferring a block of bytes or words, and generating *PWM* outputs.

PTS routine The entire microcoded response to multiple PTS interrupt requests. The PTS routine is controlled by the contents of the PTS control block.

PTS transfer The movement of a single byte or word from the source memory location to the destination memory location.

PTS vector A location in *special-purpose memory* that holds the starting address of a *PTS control block*.

QUAD-WORD An unsigned, 64-bit variable with values from 0 through 2^{64}–1. The QUAD-WORD variable is supported only as the operand for the EBMOVI instruction.

RALU Register arithmetic-logic unit. A part of the CPU that consists of the *ALU*, the *PSW*, the master *PC*, the microcode engine, a loop counter, and six registers.

reserved memory — A memory location that is reserved for factory use or for future expansion. Do not use a reserved memory location except to initialize it with FFH.

sampled inputs — All input pins, with the exception of RESET#, are sampled inputs. The input pin is sampled one state time before the read buffer is enabled. Sampling occurs during PH1 (while CLKOUT is low) and resolves the value (high or low) of the pin before it is presented to the internal bus. If the pin value changes during the sample time, the new value may or may not be recorded during the read.

RESET# is a level-sensitive input. EXTINT*x* is normally a sampled input; however, the powerdown circuitry uses EXTINT*x* as a level-sensitive input during powerdown mode.

saturation mode — Saturation occurs when the result of two positive numbers generates a negative sign bit or the result of two negative numbers generates a positive sign bit. Saturation mode prevents an underflow or overflow of the accumulated value.

set — The "1" value of a bit or the act of giving it a "1" value. See also *clear*.

SFR — Special-function register.

SHORT-INTEGER — An 8-bit, signed variable with values from -2^7 through $+2^7-1$.

sign extension — A method for converting data to a larger format by filling the upper bit positions with the value of the sign. This conversion preserves the positive or negative value of signed integers.

sink current — Current flowing **into** a device to ground. Always a positive value.

source current — Current flowing **out of** a device from V_{CC}. Always a negative value.

SP — Stack pointer.

special interrupt — Any of the three *nonmaskable interrupts* (unimplemented opcode, software trap, or NMI).

special-purpose memory — A partition of memory used for storing the *interrupt vectors*, *PTS vectors*, chip configuration bytes, and several reserved locations.

standard interrupt — Any *maskable interrupt* that is assigned to the *interrupt controller* for processing by an *interrupt service routine*.

state time (or state) — The basic time unit of the device; the combined period of the two internal timing signals, PH1 and PH2. (The internal clock generator produces PH1 and PH2 by halving the frequency of the signal on XTAL1. The rising edges of the active-high PH1 and PH2 signals generate CLKOUT, the output of the internal clock generator.) Because the device can operate at many frequencies, this manual defines time requirements in terms of *state times* rather than in specific units of time.

t — Lowercase "t" represents the period of the internal clock. For the NP, t is the reciprocal of F_{XTAL1} ($1/F_{XTAL1}$, where F_{XTAL1} is the input frequency on XTAL1). For the 80C196NU, which employs a phased-lock loop with clock multiplier circuitry, t is the reciprocal of either F_{XTAL1}, $2F_{XTAL1}$, or $4F_{XTAL1}$. The multiplier depends on the clock mode, which is controlled by the PLLEN1 and PLLEN2 input pins. (Figure 2-4 on page 2-8 illustrates the clock circuitry of the 80C196NU.)

UART — Universal asynchronous receiver and transmitter. A part of the serial I/O port.

WDT — See *watchdog timer.*

word — Any 16-bit unit of data.

WORD — An unsigned, 16-bit variable with values from 0 through $2^{16}-1$.

zero extension — A method for converting data to a larger format by filling the upper bit positions with zeros.

Index

INDEX

A

B

C

D

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z