Interrupts 5

CHAPTER 5
INTERRUPTS

A microcontroller's primary function is to provide real-time control of an instrument or device. The interrupt control circuitry within a microcontroller permits real-time events to control program flow. When an event generates an interrupt, the CPU services the interrupt before executing the next instruction. An internal peripheral, an external signal, or an instruction can request an interrupt. In the simplest case, the 8XC196KC/KD receives the request, performs the service, and returns to the task that was interrupted. This chapter describes the interrupt control circuitry, priority scheme, and timing. It also describes the three special interrupts and explains interrupt programming and control.

5.1. INTERRUPT PROCESSING

The 8XC196KC/KD provides two interrupt service options: software interrupt service routines via the Interrupt Controller and microcoded hardware interrupt processing via the Peripheral Transaction Server (PTS). You can select either option for each of the maskable interrupts. (See "Selecting Either PTS or Standard Interrupt Service" on page 5-10.) The nonmaskable interrupts (NMI, Software Trap, and Unimplemented Opcode) are always serviced by interrupt service routines. Figure 5-1 illustrates the interrupt processing flow.

5.1.1. Interrupt Controller

The Interrupt Controller services interrupts with software interrupt service routines. When the hardware detects an interrupt, it generates and executes a special interrupt call. This pushes the contents of the program counter onto the stack and then loads it with the contents of the appropriate interrupt vector. The Upper and Lower Interrupt Vectors in Special-Purpose memory (see Chapter 4, "Memory Partitions") contain the addresses of the interrupt service routines. The CPU executes the interrupt service routine. Upon completion of the service routine, the program counter is reloaded from the stack and program execution continues.

5.1.2. Peripheral Transaction Server (PTS)

The Peripheral Transaction Server (PTS) is a microcoded hardware interrupt handler. It can be used in place of a standard interrupt service routine for each of the maskable interrupts. The PTS services interrupts with less overhead; it does not modify the stack or the PSW, and it allows normal instruction flow to continue. For these reasons, the PTS can service an interrupt in the time required to execute a single instruction.

The PTS operates in five special microcoded modes that enable the PTS to complete the specific tasks in much less time than an interrupt service routine. See the "PTSCON" section on page 5-16 for a description of the PTS modes.

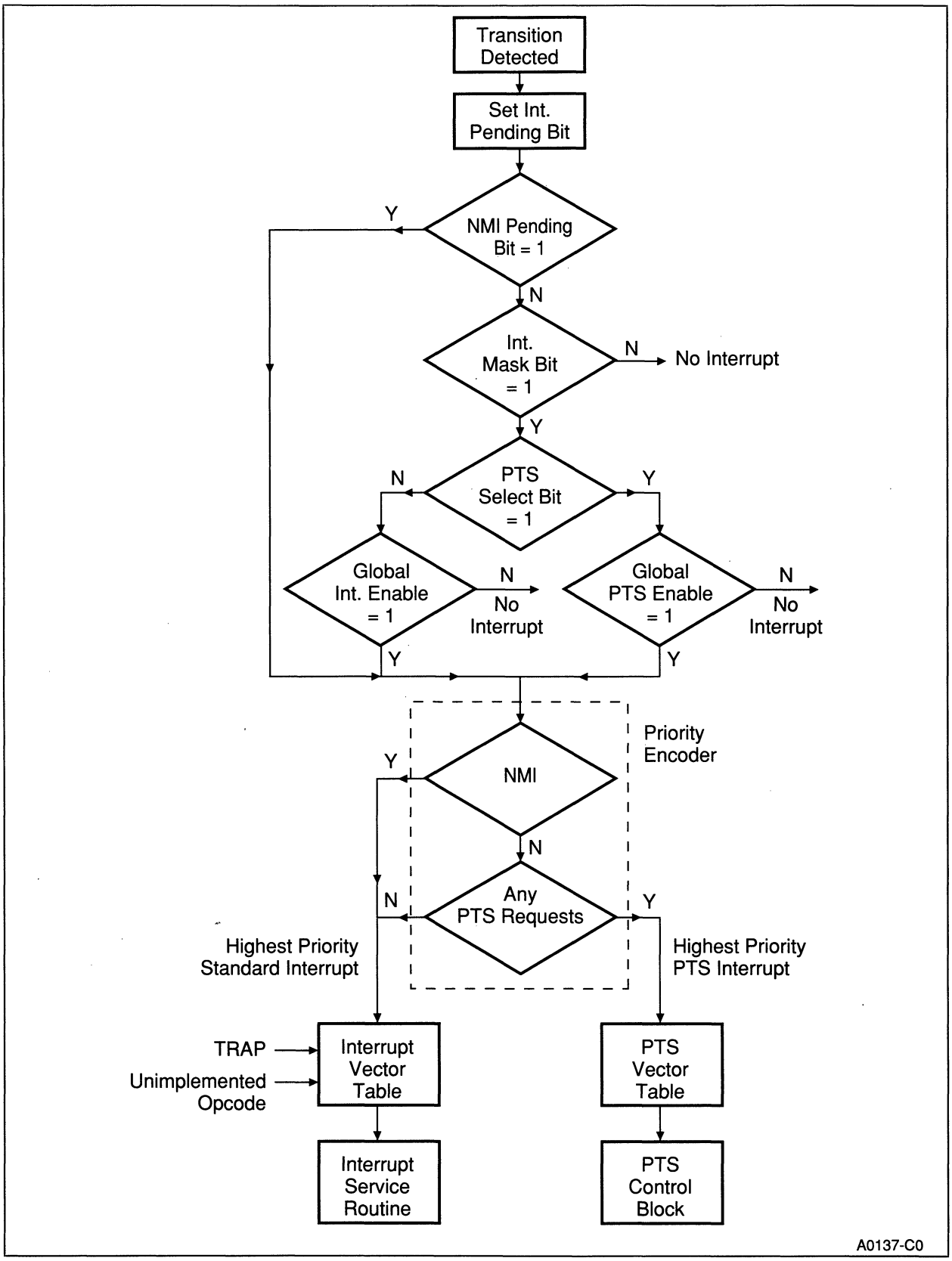

Figure 5-1. 8XC196KC/KD Flow Diagram for PTS and Standard Interrupts

Each PTS interrupt requires a block of data called the PTS Control Block (PTSCB). When a PTS interrupt occurs, the priority encoder selects the appropriate vector and fetches the PTS Control Block (PTSCB). The PTSCB determines the mode, the total number of transfers (if applicable), the total number of cycles that will execute before the PTS requires servicing, and the source and/or destination of data transfers (if applicable). Each PTS interrupt generates one PTS cycle. Figure 5-2 shows the PTS cycle flow.

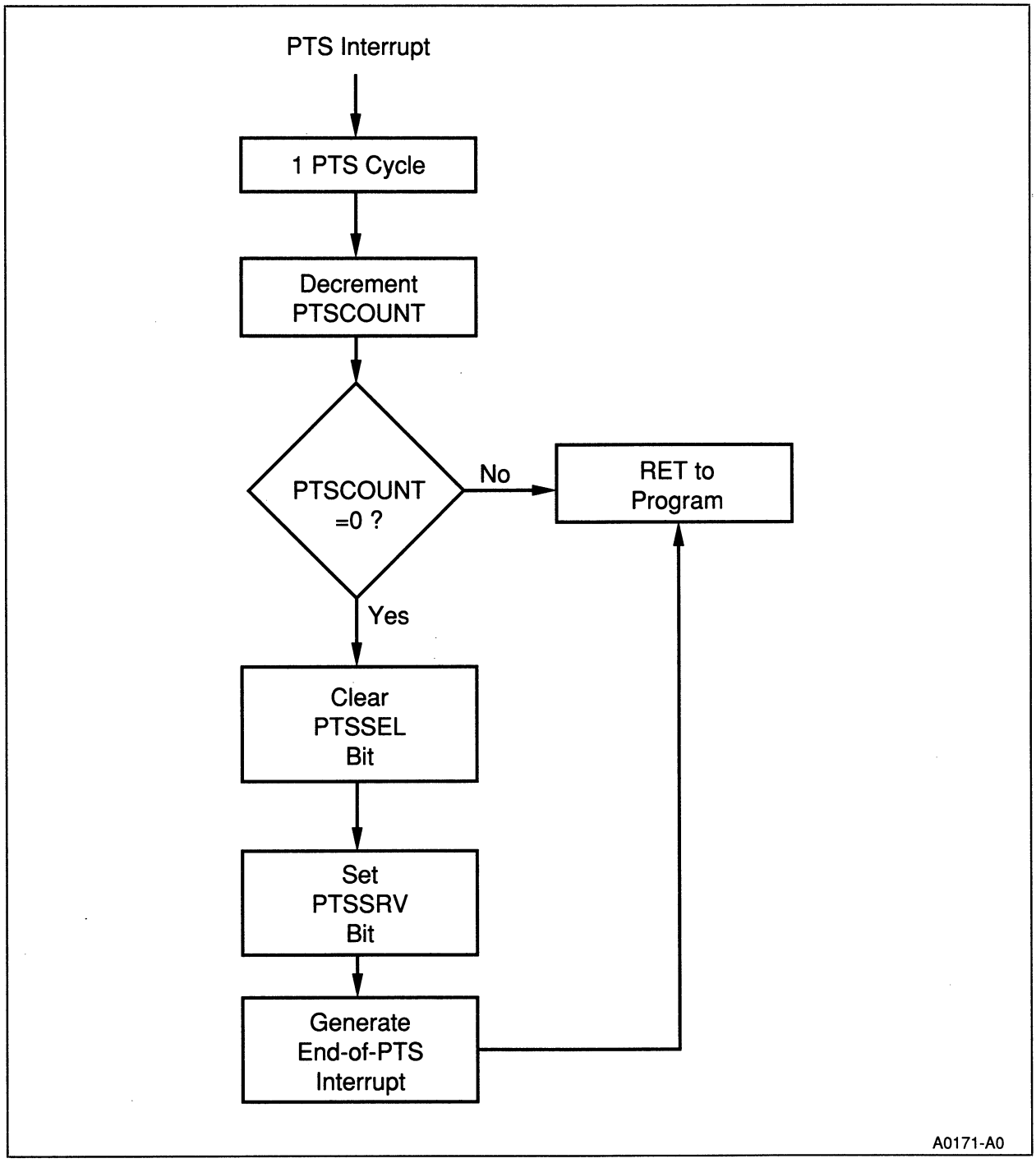

Figure 5-2. PTS Cycle Flow Diagram

5.2. INTERRUPT PRIORITIES

The Unimplemented Opcode and Software Trap interrupts are not prioritized; they go directly to the Interrupt Controller for servicing. The Interrupt Controller selects the corresponding vector location in Special-Purpose memory (see Chapter 4, "Memory Partitions"). The vector contains the starting address of the interrupt service routine.

The Priority Encoder determines the priority of all other pending interrupt requests. Table 5–1 shows the default interrupt priorities (15 is highest and 1 is lowest). NMI has the highest priority of all prioritized interrupts. If an NMI is pending, the Priority Encoder selects it as the highest priority request, and the Interrupt Controller selects the corresponding vector location in Special-Purpose memory (see Chapter 4). The vector contains the starting address of the corresponding interrupt service routine.

Any PTS interrupt request has a higher priority than all maskable standard interrupt requests. If no NMI request is pending, the Priority Encoder determines the highest priority PTS interrupt service request, and the Interrupt Controller selects the corresponding PTS vector location in Special-Purpose memory. The vector contains the starting address of the corresponding PTS control block (PTSCB).

If no NMI or PTS request is pending, the Priority Encoder determines the highest priority standard interrupt request, and the Interrupt Controller selects the corresponding vector location in Special-Purpose memory. The vector contains the starting address of the corresponding interrupt service routine.

5.2.1. Modifying Interrupt Priorities

The 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 or PTS cycle. The following code shows one way to prevent all interrupts, except EXTINT (priority 7), from interrupting a Receive interrupt service routine (priority 9).

```
SERIAL_RI_ISR:
     pusha                              ; Save PSW, INT_MASK, INT_MASK1, & WSR
     di                                 ; Disable all interrupts except EXTINT
     ldb int_mask1, #00100000B          ;
     ei                                 ; Enable interrupt servicing
                                        ;
                                        ; Service the RI interrupt
                                        ;
     popa                               ; Restore PSW, INT_MASK, INT_MASK1, &
                                        ; WSR regs
     ret
```

Note that location 2032H in the interrupt vector table would be loaded with the value of the label SERIAL_RI_ISR, and that the Receive interrupt must be enabled for this routine to execute.

Table 5-1. 8XC196KC/KD Interrupt Vector Sources, Locations, and Priorities

Number	Interrupt Vector	Source(s)	Interrupt Vector Location	PTS Vector Location	Priority (1)
Special	Unimplemented Opcode	Unimplemented Opcode	2012H	—	—
Special	Software Trap	TRAP Instruction	2010H	—	—
INT15	NMI (2)	NMI	203EH	—	15
Each of the following, maskable interrupts can be assigned to the PTS. Any PTS interrupt has priority over all other maskable interrupts.					
INT14	HSI FIFO Full	HSI FIFO Full	203CH	205CH	14
INT13	EXTINT1 (2)	P2.2	203AH	205AH	13
INT12	Timer 2 Overflow	Timer 2 Overflow	2038H	2058H	12
INT11	Timer 2 Capture (2)	Timer 2 Capture	2036H	2056H	11
INT10	HSI FIFO 4	HSI FIFO Fourth Entry	2034H	2054H	10
INT09	Receive	RI Flag (3)	2032H	2052H	9
INT08	Transmit	TI Flag (3)	2030H	2050H	8
INT07	EXTINT (2)	P2.2 or P0.7	200EH	204EH	7
INT06	Serial Port	RI Flag and TI Flag (4)	200CH	204CH	6
INT05	Software Timer	Software Timer 0–3 Timer 2 Reset A/D Conversion Start	200AH	204AH	5
INT04	HSI.0 Pin (2)	HSI.0	2008H	2048H	4
INT03	High Speed Outputs	HSO.0–HSO.5	2006H	2046H	3
INT02	HSI Data Available	HSI FIFO Full or HSI Holding Reg. Loaded	2004H	2044H	2
INT01	A/D Conversion Complete	A/D Conversion Complete	2002H	2042H	1
INT00	Timer Overflow	Timer 1 or Timer 2	2000H	2040H	0

NOTES:

1. The Unimplemented Opcode and Software Trap interrupts are not prioritized. They go directly to the Interrupt Controller for servicing. NMI has the highest priority of all prioritized interrupts. Any PTS interrupt has priority over all other maskable interrupts.
2. These interrupts can be configured to function as independent, external interrupts.
3. If the Serial interrupt is masked and the Receive and Transmit interrupts are enabled, the RI flag and TI flag generate separate Receive and Transmit interrupts. If 8096BH compatibility is not an issue, this configuration is preferred.
4. If the Receive and Transmit interrupts are masked and the Serial interrupt is enabled, both RI flag and TI flag generate a Serial Port interrupt. This configuration provides compatibility with the 8096BH.

All 8XC196KC/KD interrupt service routines are handled in the following manner:

1. After the hardware detects and prioritizes an interrupt request, it generates and executes a special 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, which is guaranteed to execute, saves the contents of the PSW, INT_MASK1, and Window Select Register (WSR) onto the stack and then clears the PSW and INT_MASK1. In addition to the arithmetic flags, the PSW contains INT_MASK register, the global interrupt enable bit (I), and the PTS enable bit (PSE). Clearing the PSW and INT_MASK1 register effectively masks all maskable interrupts, disables standard interrupt servicing, and disables the PTS. The PUSHA instruction inhibits interrupts 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 EXTINT 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_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. It is quite likely that the POPA instruction will re-enable a pending interrupt. If the Interrupt Controller serviced the pending interrupt before the RET instruction executed, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts occur at a high frequency.

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 availability of 232 bytes of RAM in the lower Register File makes this quite practical. In addition, the RAM in the upper Register File is available via *vertical windowing* (see Chapter 4).

5.3. INTERRUPT TIMING

Five external sources can interrupt the 8XC196KC/KD. The Transition Detector samples the interrupt inputs and latches the interrupt when a low-to-high transition occurs. The interrupt input must be held high for the minimum pulse width to ensure recognition. Some interrupts are sampled during Phase 1 (CLKOUT low); others are sampled during Phase 2 (CLKOUT high). Table 5-2 lists both the minimum pulse width and the sample clock phase for each external interrupt.

Table 5-2. 8XC196KC/KD Minimum Interrupt Pulse Width and Sample Clock Phase

Interrupt Source	Interrupt Vector(s)	Number	Minimum Pulse Width	Sampled During
HSI.0	HSI.0 Pin	INT04	> 2 state times	Phase 1
NMI	NMI	INT15	> 1 state time	Phase 2
P0.7	EXTINT	INT07	> 2 state times	Phase 1
P2.2	EXTINT	INT07	> 2 state times	Phase 2
	EXTINT1	INT13	> 2 state times	Phase 2
Timer 2 Capture	Timer 2 Capture	INT11	> 2 state times	Phase 2

Table 5-3 describes six instructions that always inhibit interrupts from being acknowledged until after the next instruction is executed.

Table 5-3. Instructions that Inhibit Interrupts

Instruction	Description
DI	Disables interrupts
EI	Enables interrupts following the execution of the next statement.
POPA	Pops two words off the top of 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.
POPF instruction	Pops one word off the top of the stack and places it into the PSW/INT_MASK register pair.
PUSHA	Pushes the PSW, INT_MASK, INT_MASK1, and the Window Select Register (WSR) onto the top of the stack, then clears the PSW, INT_MASK, and INT_MASK1 registers.
PUSHF	Pushes the PSW/INT_MASK register pair onto the top of the stack, then clears it.

Execution of any of the following also inhibits interrupts from being acknowledged until after the next instruction is executed:

- the signed prefix opcode (FE) for the two-byte, signed multiply and divide instructions
- the Unimplemented Opcode interrupt
- the Software Trap interrupt

5.3.1. Interrupt Latency

Latency is the total delay between the time that the interrupt is generated (not acknowledged) and the time that the 8XC196KC/KD begins executing the interrupt service routine or PTS cycle. A delay occurs between the time that the interrupt is detected and the time that it is acknowledged. An interrupt is not acknowledged until the current instruction finishes executing. If the interrupt does not occur at least four state times before the end of the current 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 generation and acknowledgment is four state times plus the execution time of the next instruction.

When a standard interrupt is acknowledged, the hardware clears the interrupt pending bit and forces a call to the address contained in the corresponding interrupt vector after completing the current instruction. The procedure that gets the vector and forces the call requires 16 state times. If the stack is in external RAM, the call requires an additional 2 state times assuming a zero-wait-state bus. When a PTS interrupt is acknowledged, it immediately vectors to the PTSCB and begins executing the PTS cycle.

5.3.2. Calculating Latency

The maximum latency occurs when the interrupt occurs too late for acknowledgment following the current instruction. The following worst-case calculation assumes that the current instruction does not inhibit interrupts (see Table 5-3). To calculate latency, add the following terms:

- Four state times to allow the current instruction to finish
- The total number of state times for the next instruction; the longest instruction (NORML) takes 39 state times
- The response time (16 state times for an internal stack or 18 for an external stack) for standard interrupts only

5.3.2.1. STANDARD INTERRUPT LATENCY

The maximum delay for a standard interrupt is 61 state times (4 + 39 + 18). This delay time does not include time needed to execute the first instruction in the interrupt service routine. Figure 5-3 illustrates an example of this worst-case scenario.

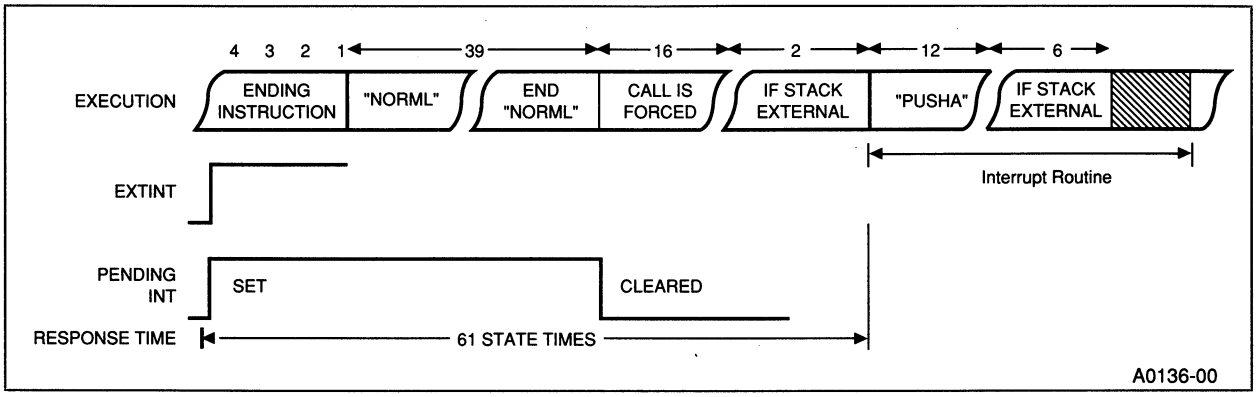

Figure 5-3. Standard Interrupt Response Time

5.3.2.2. PTS INTERRUPT LATENCY

The maximum delay for a PTS interrupt is 43 state times (4 + 39). This delay time does not include the added delay if the PTS is disabled (PSW.2 clear) or if a higher priority PTS request is being serviced. See Table A-11 in Appendix A for the PTS cycle execution times.

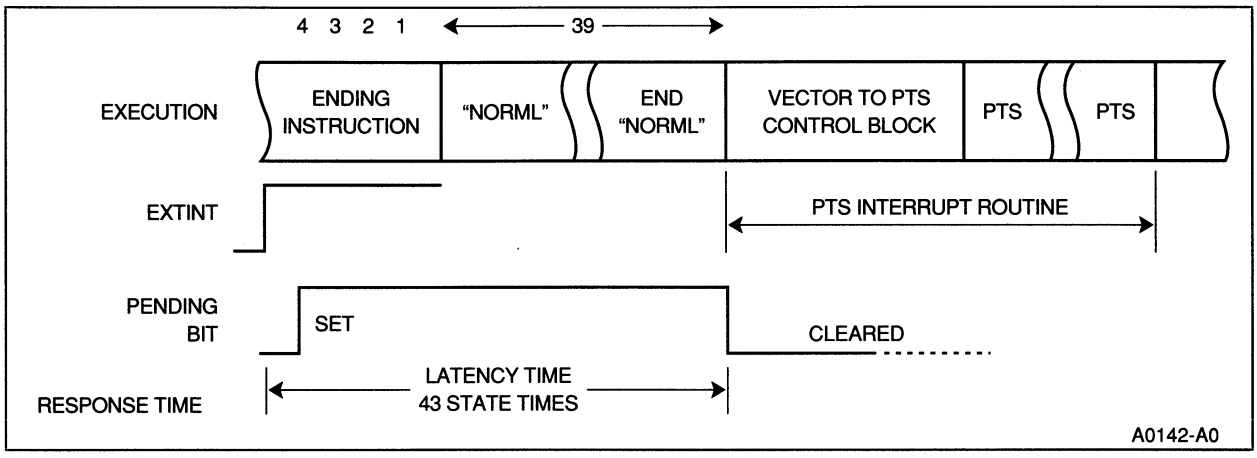

Figure 5-4. PTS Interrupt Response Time

5.4. SPECIAL INTERRUPTS

The 8XC196KC/KD supports three special interrupts: Unimplemented Opcode, Software Trap, and NMI. These interrupts are not affected by the interrupt enable bit (I) in the PSW (PSW.1), 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 Intel development tools.

5.4.1. Unimplemented Opcode

If the CPU attempts to execute an unimplemented opcode, an indirect vector through location 2012H 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 interrupts from being acknowledged until after the next instruction is executed.

5.4.2. Software Trap

The TRAP instruction (opcode 0F7H) causes an interrupt call that is vectored through location 2010H. The TRAP instruction provides a single-instruction interrupt that is useful when debugging software or generating software interrupts. The TRAP instruction prevents other interrupts from being acknowledged until after the next instruction is executed.

5.4.3. 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 203EH. The NMI interrupt is sampled during Phase 2 (CLKOUT high) and is latched internally. If the pin is held high, multiple interrupts will not occur. If your system does not use the NMI interrupt, ground the NMI pin to prevent spurious interrupts.

For design symmetry with the INT_PEND1 register, an NMI mask bit exists in the INT_MASK1 register. However, the mask bit has no function; NMI is enabled for both NMI_MASK set and NMI_MASK cleared. To ensure compatibility with future products, always write zero to the NMI mask bit.

The NMI on the 8096BH vectors directly to location 0000H. For compatibility with 8096BH software that uses the NMI, load 0000H into location 203EH.

5.5. PROGRAMMING THE INTERRUPTS

Table 5-4 lists the programmable registers that affect the performance and function of the Interrupt Controller and PTS. Refer to Appendix C for detailed descriptions of these registers.

5.5.1. Selecting Either PTS or Standard Interrupt Service

The PTS Select register (PTSSEL) selects either a PTS cycle or a standard software interrupt service routine for each of the fifteen maskable interrupt requests. Setting a bit selects a PTS cycle; clearing a bit selects a standard interrupt service routine. See Appendix C for a detailed description of the PTSSEL register.

Table 5-4. Interrupt and PTS Control and Status Registers

Register Mnemonic	Register Name	Description
INT_MASK INT_MASK1	Interrupt Mask Registers	These registers enable/disable each maskable interrupt (that is, each interrupt except Unimplemented Opcode, Software Trap, and NMI.)
INT_PEND INT_PEND1	Interrupt Pending Registers	The bits in this register are set by hardware to indicate that an interrupt is pending.
IOC1	Input/Output Control Register 1	This register selects the source of the INT00, INT02, and INT07 interrupts.
IOS1	Input/Output Status Register 1	This register contains flags that indicate which events triggered interrupts.
PSW	Program 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.
PTSSEL	PTS Select Register	This register selects either a PTS cycle or a standard interrupt service routine for each of the fifteen maskable interrupt requests.
PTSSRV	PTS Service Register	The bits in this register are set by hardware to request an end-of-PTS interrupt.

5.5.2. Enabling PTS Interrupts

After you assign an interrupt to the PTS, you must enable both the PTS and the individual interrupt. The PTS enable (PSE) bit in the Program Status Word (PSW.2) globally enables or disables the PTS. The EPTS instruction sets the bit, which enables the PTS. The DPTS instruction clears the bit, which disables the PTS. The bits in INT_MASK and INT_MASK1 individually enable or disable the PTS interrupts (see Table 5-5).

5.5.3. Enabling Standard Interrupts

When you assign an interrupt to a standard software service routine, you must enable both the servicing of the interrupt and the individual interrupt. The global interrupt enable (I) bit in the Program Status Word (PSW.1) globally enables or disables the servicing of all maskable interrupts. The EI instruction sets the bit, which enables interrupt servicing. The DI instruction clears the bit, which disables interrupt servicing. The bits in INT_MASK and INT_MASK1 individually enable or disable the interrupts (see Table 5-5). Interrupts that occur while interrupt servicing is globally disabled (PSW.1 cleared) are held in the interrupt pending registers.

Table 5-5. Standard Interrupt Sources, Vectors, and Register Bits

Interrupt Source	Interrupt Vector(s)	Number	Interrupt Enabled by Setting (1)	Source Selected by (2)
A/D Conversion Complete	A/D Conversion Complete	INT01	INT_MASK.1	—
A/D Conversion Start	Software Timer	INT05	INT_MASK.5	—
HSI FIFO Fourth Entry	HSI FIFO 4	INT10	INT_MASK1.2	—
HSI FIFO Full	HSI FIFO Full	INT14	INT_MASK1.6	—
	HSI Data Available	INT02	INT_MASK.2	IOC1.7 = 1
HSI Holding Reg. Loaded	HSI Data Available	INT02	INT_MASK.2	IOC1.7 = 0
HSI.0	HSI.0 Pin	INT04	INT_MASK.4	—
HSO.0–HSO.5	High Speed Outputs	INT03	INT_MASK.3	—
NMI	NMI	INT15	—	—
P0.7	EXTINT	INT07	INT_MASK.7	IOC1.1 = 1
P2.2	EXTINT	INT07	INT_MASK.7	IOC1.1 = 0
	EXTINT1	INT13	INT_MASK1.5	—
RI Flag	Receive	INT09	INT_MASK1.1	—
	Serial Port	INT06	INT_MASK.6	—
Software Timers 0–3	Software Timer	INT05	INT_MASK.5	—
TI Flag	Transmit	INT08	INT_MASK1.0	—
	Serial Port	INT06	INT_MASK.6	—
Timer 1 Overflow	Timer Overflow	INT00	INT_MASK.0	IOC1.2 = 1
Timer 2 Capture	Timer 2 Capture	INT11	INT_MASK1.3	—
Timer 2 Overflow	Timer Overflow	INT00	INT_MASK.0	IOC1.3 = 1
	Timer 2 Overflow	INT12	INT_MASK1.4	—
Timer 2 Reset	Software Timer	INT05	INT_MASK.5	—

NOTES:

1. This column lists, for each interrupt source, the mask bit that must be set to enable the interrupt. Five interrupt sources can each generate two different interrupts — an 8096BH-compatible interrupt and a new, separate interrupt (HSI FIFO Full, EXTINT1, Receive, Transmit, Timer 2 Overflow). In all cases, only one interrupt should be enabled for each source. (That is, the mask bit should be set for only one of the two possible interrupts).
2. Three of the 8096BH-compatible interrupts (HSI Data Available, EXTINT, and Timer Overflow) can be generated by either of two sources. This column shows the IOC1 register bit and value that selects each source.

5.5.4. Selecting Interrupt Sources

Five interrupt sources can each generate two different interrupts — an 8096BH-compatible interrupt and a new, separate interrupt (HSI FIFO Full, EXTINT1, Receive, Transmit, Timer 2 Overflow). In all cases, only one interrupt should be enabled for each source. (That is, the mask bit should be set for only one of the two possible interrupts). Figure 5-5 shows the interrupt sources for each interrupt vector. Table 5-5 lists the IOC1 register bit and value that selects each source.

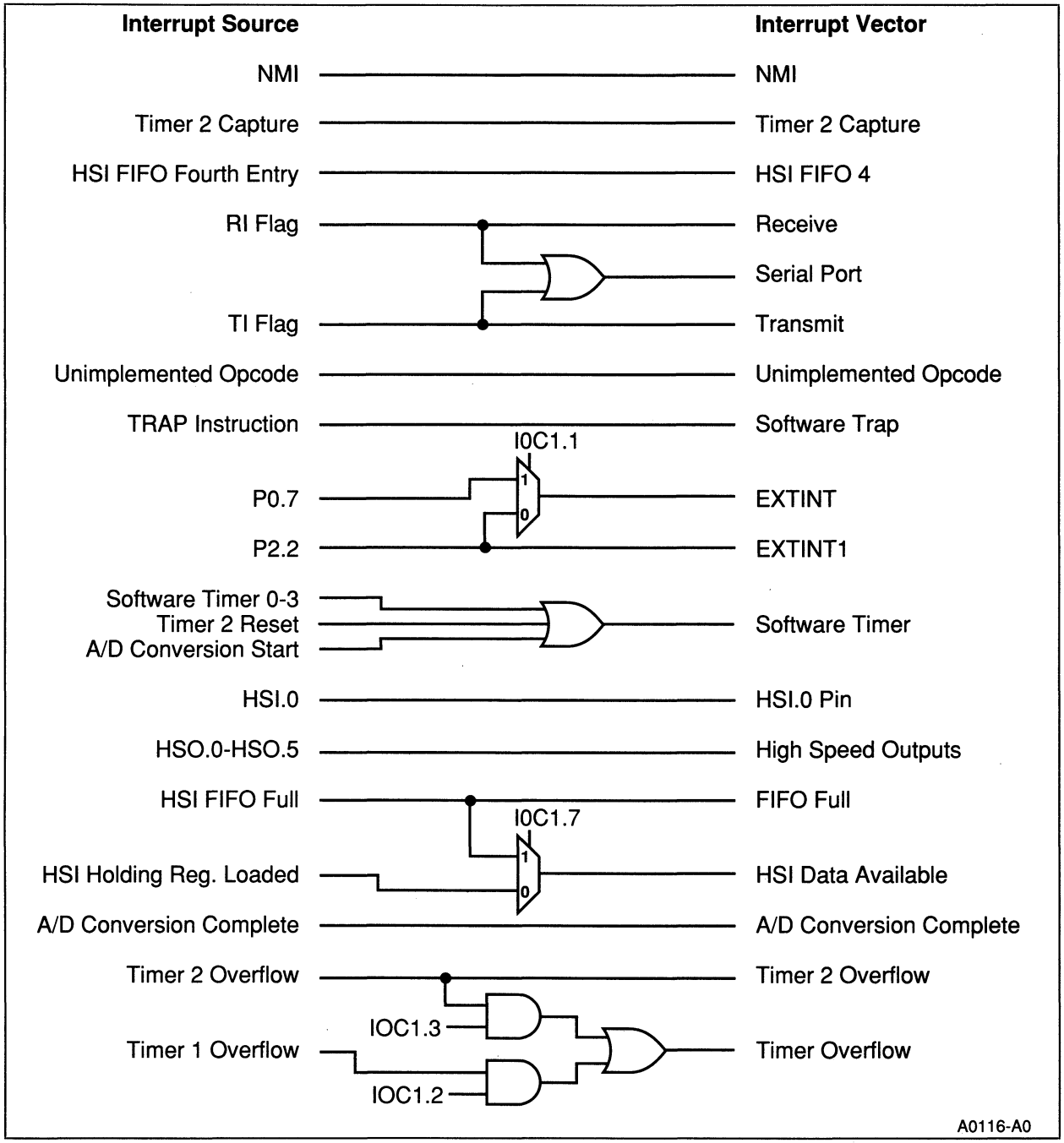

Figure 5-5. Interrupt Sources

5.5.5. Interrupt Mask Registers

The interrupt mask registers, INT_MASK and INT_MASK1, enable or disable (mask) individual interrupts. With the exception of the Nonmaskable Interrupt (NMI) bit (INT_MASK1.7), setting a bit enables the corresponding interrupt source; clearing a bit disables the source. When the device is reset, the interrupt mask registers are cleared (disabling interrupts).

For design symmetry with the INT_PEND1 register, an NMI mask bit exists in the INT_MASK1 register. However, the mask bit has no function; NMI is enabled for both NMI_MASK set and NMI_MASK cleared. To ensure compatibility with future products, always write zero to the NMI mask bit.

When the device is reset, the interrupt mask registers are cleared (disabling interrupts).

5.5.6. Interrupt Pending Registers

When the Transition Detector detects an interrupt, it sets the corresponding bit in the INT_PEND or INT_PEND1 register. 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 (standard interrupts) or PTSCB (PTS interrupts). 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.

Care should be taken in writing code that modifies these registers. For example, an instruction sequence that clears a pending bit could result in an interrupt being acknowledged after the sequence begins but before the bit is actually cleared. In this case a five-state-time *partial* interrupt cycle occurs. That is, the interrupt process begins, but never jumps to the interrupt service routine. This time delay can be avoided by making the code inseparable, in the sense that an interrupt will not be acknowledged while the code is executing. The easiest way to do this is to use the logical instructions in the two- or three-operand format, for example:

```
ANDB INT_PEND, #01111111B      ; Clears the EXTINT interrupt
ORB INT_PEND, #10000000B       ; Sets the EXTINT interrupt
```

The 8XC196KC/KD does not acknowledge interrupts during execution of these "read-modify-write" instructions.

The PTSSRV register holds requests for "end-of-PTS" interrupts. End-of-PTS interrupts indicate that the PTS needs servicing. The end-of-PTS interrupt service routine should reinitialize the PTSCB and set the appropriate PTSSEL bit to re-enable PTS interrupt service.

5.6. PTS CONTROL BLOCKS

Each PTS interrupt requires a block of data called the PTS Control Block (PTSCB). The PTSCB determines the PTS mode, the number of PTS cycles, and the address of the source and destination of data transfers. You must set up the PTSCB for each interrupt source before enabling the PTS interrupts. Each PTSCB requires eight data bytes in register RAM. The address of the first (lowest) byte is stored in the PTS Vector table in Special-Purpose memory (see Chapter 4, "Memory Partitions"). Write the first byte into an address evenly divisible by 8 (quad-word boundary). Figure 5-6 shows the PTSCB for each PTS mode. Unused PTSCB bytes can be used as extra RAM.

Single Transfer	Block Transfer	A/D Scan Mode	HSO Mode	HSI Mode
Unused	Unused	Unused	Unused	Unused
Unused	PTSBLOCK	Unused	PTSBLOCK	PTSBLOCK
PTSDST (HI)	PTSDST (HI)	PTSREG (HI)	Unused	Unused
PTSDST (LO)	PTSDST (LO)	PTSREG (LO)	Unused	Unused
PTSSRC (HI)	PTSSRC (HI)	PTS_S/D (HI)	PTSSRC (HI)	PTSDST (HI)
PTSSRC (LO)	PTSSRC (LO)	PTS_S/D (LO)	PTSSRC (LO)	PTSDST (LO)
PTSCON	PTSCON	PTSCON	PTSCON	PTSCON
PTSCOUNT	PTSCOUNT	PTSCOUNT	PTSCOUNT	PTSCOUNT

Figure 5-6. PTS Control Blocks

5.6.1. PTSCOUNT Register

The first location of each PTSCB is always the PTSCOUNT register. PTSCOUNT defines the number of PTS cycles to be executed consecutively without software intervention. Since PTSCOUNT is an 8-bit value, the maximum number of cycles is 256. PTSCOUNT is decremented at the end of each PTS cycle. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.

5.6.1.1. END-OF-PTS INTERRUPTS

An end-of-PTS interrupt is a 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 Transmit interrupt if PTSSEL.8 is set. The interrupt vectors through 2050H, but the corresponding end-of-PTS interrupt vectors through 2030H, the standard Transmit interrupt vector. When the end-of-PTS interrupt vectors to the interrupt service routine, hardware clears the PTSSRV bit. The interrupt service routine must set the PTSSEL bit to re-enable PTS service for the interrupt.

5.6.2. PTSCON Register

The second location of each PTSCB is always the PTSCON register. Three bits of the PTSCON register determine the PTS mode: Single Transfer, Block Transfer, A/D Scan, HSO, or HSI (see Table 5-6). The PTS mode defines the functions of the other bits; see Table 5-8 for Single and Block Transfer modes and Table 5-8. PTSCON has one configuration for the Single and Block Transfer modes (see Figure 5-7) and one for the A/D Scan, HSO, and HSI modes (see Figure 5-8). See Appendix C for a detailed description of the contents of PTSCON during each PTS mode. See Table A-11 in Appendix A for cycle execution time for each PTS mode.

Table 5-6. PTS Mode Select

Bit 7	Bit 6	Bit 5	Selected Mode
0	0	0	Single Transfer
0	0	1	HSI Mode
0	1	0	N/A
0	1	1	HSO Mode
1	0	0	Block Transfer
1	0	1	N/A
1	1	0	A/D Scan
1	1	1	N/A

Table 5-7. PTSCON Bits 0–4 (Single and Block Transfer Modes)

Bit Number	Bit Mnemonic	Bit Name	Description
0	DI	PTSDST Auto-Increment	Setting this bit causes the PTS destination register to increment at the end of each PTS cycle.
1	SI	PTSSRC Auto-Increment	Setting this bit causes the PTS source register to increment at the end of each PTS cycle.
2	DU	Update PTSDST	Setting this bit causes the PTSDST register to retain its final value at the end of a PTS cycle. Clearing it causes the register revert to the value that existed at the beginning of the PTS cycle.
3	SU	Update PTSSRC	Setting this bit causes the PTSSRC register to retain its final value at the end of a PTS cycle. Clearing it causes the register to revert to the value that existed at the beginning of the PTS cycle.
4	BW	Byte/Word Transfer	Setting this bit specifies a byte transfer. Clearing it specifies a word transfer.

Table 5-8. PTSCON Bit 3 (A/D Scan, HSI, and HSO Modes)

Bit Number	Bit Mnemonic	Bit Name	Description
3	UPDT	Update Register	Setting this bit causes the associated register(s) to be loaded with the value that exists at the end of a PTS cycle. Clearing it causes the register(s) to be loaded with the value that existed at the beginning of the PTS cycle. **Mode Register** A/D PTS_S/D HSI PTSDST HSO PTSSRC

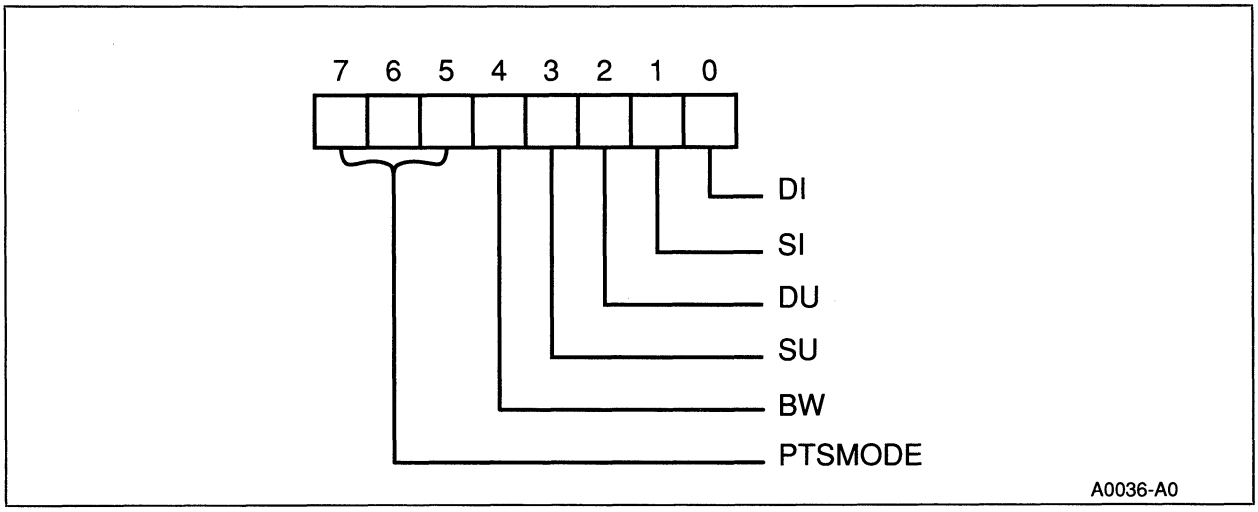

Figure 5-7. PTSCON (Single and Block Transfer Modes)

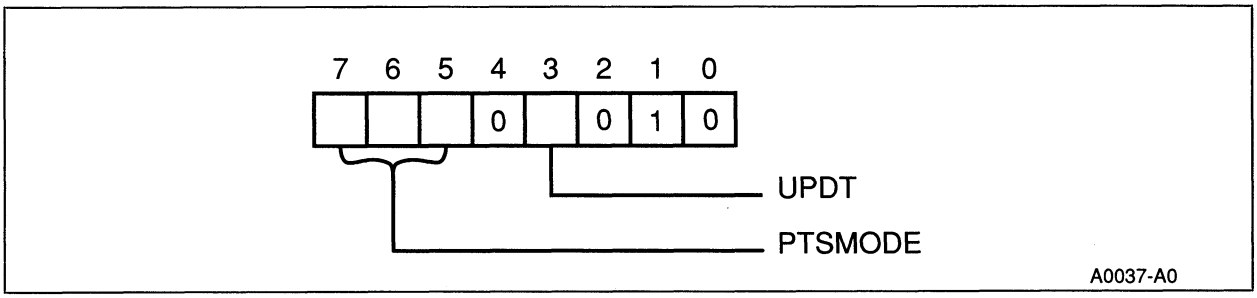

Figure 5-8. PTSCON (A/D Scan, HSI, and HSO Modes)

5.7. SINGLE TRANSFER MODE

In the Single Transfer mode, each PTS cycle transfers 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 port interrupts. The PTSCOUNT register specifies the number of transfers (each transfer is one PTS cycle). The PTS moves the byte or word from the location pointed to by the source register (PTSSRC) to the location pointed to by the destination register (PTSDST).

PTSSRC and PTSDST may point to any memory location; however, they must point to an even address if word transfers are selected. Setting the auto-increment and update bits causes the PTS to increment the source (if SI and SU are set) and/or destination (if DI and DU are set) address at the end of each PTS cycle. The address increments by one if byte transfers are selected or by two if word transfers are selected. In Single Transfer mode, each pair of auto-increment and update bits must both be either set or cleared. Programming the increment and update bits to (0,1) or (1,0) selects an invalid mode. PTSSRC and PTSDST can be incremented (and updated) independently of each other.

5.7.1. Single Transfer Mode Example

The following PTSCB defines nine PTS cycles (see Table 5-9). 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 6000H–600FH, and an end-of-PTS interrupt is generated.

Table 5-9. Single Transfer Mode PTSCB

Unused
Unused
PTSDST (HI) = 60H
PTSDST (LO) = 00H
PTSSRC (HI) = 00H
PTSSRC (LO) = 20H
PTSCON = 15H (DI, DU, & BW = 1)
PTSCOUNT = 09H

5.8. BLOCK TRANSFER MODE

In Block Transfer mode, the PTS moves a block of data from one memory location to another. The PTSBLOCK register specifies the number of bytes or words in each block (n = 1–32). The PTS moves the block of bytes or words from the location pointed to by the source register (PTSSRC) to the location pointed to by the destination register (PTSDST).

PTSSRC and PTSDST may point to any memory location; however, they must point to an even address if word transfers are selected. Setting the auto-increment bits in the PTSCON register, causes the PTS to increment the source (SI set) and/or destination (DI set) address at the end of each PTS transfer. If the update bit is also set, the incremented address is saved in the PTSSRC (SU set) or PTSDST (DU set) register after each PTS cycle. Setting both the increment and update bits means that the source and/or destination address will be incremented after each cycle. The registers increment by one if byte transfers are selected or by two if word transfers are selected. The increment and update features may be selected independently (unlike in Single Transfer Mode).

In this mode, it is important to differentiate between a *PTS transfer* and a *PTS cycle*. A *PTS transfer* is the movement of a single byte or word from the source to the destination. A *PTS cycle* consists of the transfer of an entire block of bytes or words. Because a PTS cycle is uninterruptable, the Block Transfer mode can create long interrupt latency. The worst-case latency could be as high as 500 states. This worst-case latency assumes a block transfer of 32 words from one external memory location to another using an 8-bit bus with no wait states. See Table A-11 in Appendix A for PTS cycle execution times.

5.8.1. Block Transfer Mode Example

The PTSCB in Table 5-10 defines three PTS cycles that will each transfer the bytes in memory locations 20H–24H to one of the following blocks: 6000H–6004H, 6005H–6009H, or 600AH–600EH. Each PTS cycle requires a burst of five transfers. The source and destination are incremented after each transfer, but only the destination is updated after each cycle. The first byte of each cycle is always read from location 20H.

Table 5-10. Block Transfer Mode PTSCB

Unused
PTSBLOCK = 05H
PTSDST (HI) = 60H
PTSDST (LO) = 00H
PTSSRC (HI) = 00H
PTSSRC (LO) = 20H
PTSCON = 97H (DI, SI, DU, BW = 1)
PTSCOUNT = 03H

5.9. A/D SCAN MODE

In the A/D Scan mode, the PTS causes the A/D converter to perform multiple conversions on one or more channels and then stores the results. To use the A/D Scan mode, you must first set up a command/data table in memory (see Table 5-11). The command/data table contains A/D commands that are interleaved with blank memory locations. The PTS stores the conversion results in these blank locations.

To initiate A/D Scan mode, enable the A/D Conversion Complete interrupt and assign it to the PTS, then have software start the first conversion. When the A/D finishes the first conversion and generates an A/D Conversion Complete interrupt, the PTS cycle is initiated.

During each PTS cycle, the PTS stores the results from the previous conversion and then executes the next conversion command. Since the conversion results are not stored until the next PTS cycle, the last command location should contain all zeros to prevent a final conversion from starting. Typically, the A/D commands are loaded into the table from an external ROM. Only the amount of available memory limits the table size; it can reside in internal or external RAM.

Table 5-11. A/D Scan Mode Command/Data Table

XXX + 0AH	A/D Result 2	
XXX + 8H	Unused	A/D Command 3
XXX + 6H	A/D Result 1	
XXX + 4H	Unused	A/D Command 2
XXX + 2H	A/D Result 0*	
XXX	Unused	A/D Command 1

*Result of the A/D conversion that initiates the PTS cycle.

In A/D Scan mode, the PTSCOUNT specifies the total number of A/D conversion cycles. The PTS_S/D register points to the table of conversion commands and results. Setting the UPDT bit in the PTSCON register (PTSCON.3) causes the PTS_S/D register to retain its final value at the end of the PTS cycle. Clearing it causes the register to revert to the value that existed at the beginning of the PTS cycle. PTS_REG points to address 02H in HWindow 0. When read, this location contains the AD_RESULT register; when written, it contains the AD_COMMAND register. The A/D Scan mode also uses two temporary registers that are inaccessible to the user.

5.9.1. PTS Cycles in A/D Scan Mode

Software must start the first A/D conversion. The A/D Conversion Complete interrupt initiates the PTS cycle. The following actions occur after the PTS cycle begins:

1. The PTS reads the first command, stores it in a temporary location, and then increments the PTS_S/D register twice. PTS_S/D now points to the first blank location in the command/data table (address xxx + 2).

2. The PTS reads the AD_RESULT register, stores the results of the first conversion into location xxx + 2 in the command/data table, and increments the PTS_S/D register twice. PTS_S/D now points to xxx + 4.

3. The PTS loads the command from the temporary location into the AD_COMMAND register. This starts the next A/D conversion cycle.

4. If UPDT (PTSCON.3) is clear, the PTS_S/D register is reinitialized to its original value. The next cycle will use the same command and overwrite previous data. If UPDT is set, the PTS saves the new contents of PTS_S/D and it points to the next command.

5. PTSCOUNT is decremented and the CPU returns to regular program execution. When PTSCOUNT reaches zero, hardware clears the corresponding PTSSEL bit and sets the PTSSRV bit, which requests the end-of-PTS interrupt.

When the conversion started by the PTS cycle completes and the A/D generates the A/D Conversion Complete interrupt, a new PTS cycle begins. Steps 1–5 repeat.

Because the lower six bits of the AD_RESULT register contain status information, the end-of-PTS interrupt service routine could shift the results data to the right six times to leave only the conversion results in the memory locations.

5.9.2. A/D Scan Mode Example 1

The command/data table shown in Table 5-12 sets up a series of A/D conversions, beginning with channel 7 and ending with channel 0. Each table entry is a word (two bytes). Table 5-13 shows the corresponding PTSCB.

Software starts a conversion on Channel 7. Upon completion of the conversion, the A/D Conversion Complete interrupt initiates the first PTS cycle. Step 1 stores the Channel 6 command in a temporary location and increments PTS_S/D to 102H. Step 2 stores the result of the Channel 7 conversion in location 102H and increments PTS_S/D to 104H. Step 3 loads the Channel 6 command from the temporary location into the AD_COMMAND register to start the next conversion. Step 4 updates PTS_S/D (PTS_S/D points to 104H) and step 5 decrements PTSCOUNT to 7. The next cycle begins by storing the Channel 5 command in the temporary location. During the eighth cycle (PTSCOUNT = 1) the dummy command is loaded into the AD_COMMAND register and no conversion is performed. PTSCOUNT is decremented to zero and the end-of-PTS interrupt is requested.

Table 5-12. Command/Data Table (Example 1)

Address	Contents
11EH	AD_RESULT for ACH0
11CH	0000H (Dummy command)
11AH	AD_RESULT for ACH1
118H	AD_COMMAND for ACH0
116H	AD_RESULT for ACH2
114H	AD_COMMAND for ACH1
112H	AD_RESULT for ACH3
110H	AD_COMMAND for ACH2
10EH	AD_RESULT for ACH4
10CH	AD_COMMAND for ACH3
10AH	AD_RESULT for ACH5
108H	AD_COMMAND for ACH4
106H	AD_RESULT for ACH6
104H	AD_COMMAND for ACH5
102H	AD_RESULT for ACH7
100H	AD_COMMAND for ACH6

Table 5-13. A/D Scan Mode PTSCB (Example 1)

Unused
Unused
PTS_REG (HI) = 00H
PTS_REG (LO) = 02H
PTS_S/D (HI) = 01H
PTS_S/D (LO) = 00H
PTSCON = CAH (UPDT = 1)
PTSCOUNT = 08H

5.9.3. A/D Scan Mode Example 2

Table 5-14 sets up a series of ten PTS cycles that each read a single A/D channel and store the result in a single location (102H). UPDT is cleared so that original contents of PTS_S/D are restored after the cycle. The command/data table is shown in Table 5-15.

Table 5-14. A/D Scan Mode PTSCB (Example 2)

Unused
Unused
PTS_REG (HI) = 00H
PTS_REG (LO) = 02H
PTS_S/D (HI) = 01H
PTS_S/D (LO) = 00H
PTSCON = C2H (UPDT = 0)
PTSCOUNT = 0AH

Table 5-15. Command/DataTable (Example 2)

Address	Contents
102H	AD_RESULT for ACH*x*
100H	AD_COMMAND for ACH*x*

Software starts a conversion on Channel *x*. The first PTS cycle begins when the conversion is finished and the A/D Conversion Complete interrupt is generated. The PTS stores the conversion results in location 102H and then copies the conversion command from location 100H to the AD_COMMAND register. The CPU can process or move the conversion results data from the table before the next conversion completes and a new PTS cycle begins. When the next cycle begins, PTS_S/D again points to 100H. The conversion results are written to location 102H and the command at location 100H is re-executed.

5.10. HSI MODE

In HSI mode, the PTS dumps the contents of HSI FIFO to a table in either internal or external memory. The PTSDST register contains the address of the table.

Any HSI interrupt can be used to trigger the PTS cycle. The PTSBLOCK register specifies how many HSI FIFO blocks (n = 1–7) will be transferred to the memory table during each PTS cycle. Enter a value that corresponds with the interrupt that generates the PTS cycle. For example, the fourth FIFO entry can cause the HSI to generate the HSI FIFO 4 interrupt (INT10). If this interrupt was used to initiate the PTS cycle, then you would write 4 into the PTSBLOCK register so that the four FIFO entries would be dumped to the table.

FIFO data is accessed by reading the HSI_STATUS register first and then reading the HSI_TIME register. The HSI_STATUS register contains the event status bits and the current HSI pin states. The HSI_TIME register contains the time, with respect to the Timer 1 count value, at which the HSI event was triggered. See Chapter 8, "High-Speed Input/Output Unit," for a detailed description of the HSI module and Appendix C, "8XC196KC/KD Registers," for a detailed description of the HSI registers.

Each PTS transfer moves the HSI_STATUS and HSI_TIME registers into consecutive words in memory (see Table 5-16). Setting the UPDT bit (PTSCON.3) causes the PTSDST register to retain its final value at the end of the PTS cycle.

Table 5-16. HSI Mode PTS Table

XXX + 0AH	HSI_TIME_2	
XXX + 8H	0FFH	HSI_STATUS_2
XXX + 6H	HSI_TIME_1	
XXX + 4H	0FFH	HSI_STATUS_1
XXX + 2H	HSI_TIME_0	
XXX	0FFH	HSI_STATUS_0

5.10.1. HSI Mode Example

The PTSCB in Table 5-17 defines ten PTS cycles that will each transfer seven blocks of HSI_STATUS/HSI_TIME data from the HSI FIFO to a table starting at memory location 100H. The destination address is incremented after each transfer and updated with the new value after each cycle.

Table 5-17. HSI Mode PTSCB

Unused
PTSBLOCK = 07H
Unused
Unused
PTSDST (HI) = 01H
PTSDST (LO) = 00H
PTSCON = 2AH (UPDT = 1)
PTSCOUNT = 0AH

5.11. HSO MODE

In HSO mode, the PTS loads the HSO CAM file from a table located in either internal or external memory (see Table 5-18). The HSO mode is similar to the HSI mode, except that the PTS moves the data from the table to the HSO CAM file instead of vice versa. The PTSSRC register contains the address of the table.

Table 5-18. HSO Mode PTS Table

XXX + 0AH	HSO_TIME_2	
XXX + 8H	Unused	HSO_COMMAND_2
XXX + 6H	HSO_TIME_1	
XXX + 4H	Unused	HSO_COMMAND_1
XXX + 2H	HSO_TIME_0	
XXX	Unused	HSO_COMMAND_0

Each CAM register is 24 bits wide. Sixteen bits identify when, with respect to Timer 1 or Timer 2, the action should occur. The remaining 8 bits define the action. Load this information into the memory table in the format shown in Table 5-18. Each PTS cycle transfers data to the HSO_COMMAND and HSO_TIME registers. The data is transferred from the registers into the HSO holding register. The command is held in the holding register until there is an empty CAM register, at which time the command enters the CAM. See Chapter 8, "High-Speed Input/Output Unit" for more information about the CAM file.

Any HSO interrupt can be used to trigger the PTS cycle. The PTSBLOCK register specifies how many HSO entries (n = 1–8) will be transferred from the memory table during each PTS cycle. Setting the UPDT bit (PTSCON.3) causes the PTSSRC register to retain its final value at the end of the PTS cycle. See Chapter 8, "High-Speed Input/Output Unit," for a detailed description of the HSO module and Appendix C, "8XC196KC/KD Registers," for information about the HSO registers.

5.11.1. HSO Mode Example

The PTSCB in Table 5-19 defines ten PTS cycles that will each transfer eight blocks of HSO_COMMAND/HSO_TIME data to the HSO from the table starting at memory location 100H. The source address is incremented after each transfer and updated with the new value after each cycle.

Table 5-19. HSO Mode PTSCB

Unused
PTSBLOCK = 08H
Unused
Unused
PTSSRC (HI) = 01H
PTSSRC (LO) = 00H
PTSCON = 6AH (UPDT = 1)
PTSCOUNT = 0AH