intel®

AP-717

APPLICATION NOTE

Migrating from the 8XC196NP or 8XC196NU to the 80296SA

January 1997

Order Number: 272730-002

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

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

or call 1-800-879-4683

CONTENTS

FIGURES

TABLES

1.0 INTRODUCTION

The 80296SA controller is the latest addition to the MCS® 96 controller family. The 80296SA was the first core redesign since the 8*x*9*x* controller was moved from an HMOS process to a CMOS process in 1986, creating the 8*x*C196KB. The core redesign means that the 80296SA microcontroller started from a blank drawing board, enhancing its performance while maintaining binary code compatibility with earlier MCS 96 controllers. The 80296SA is pin compatible with the 8*x*C196NP and 8*x*C196NU, so you can place the 80296SA into a socket designed for its predecessors. The 80296SA has a four-stage pipelined architecture: fetch, decode, read-execute, and execute-write stages.

The 80296SA exhibits improved math performance over the previous architectures, making it more suitable for embedded digital signal processing. It can perform 12.5 DSP MIPS and 16 general-purpose MIPS. The DSP MIPS value is calculated using the multiply-accumulate (MAC) execution time on register-to-register operations (two state times). The general-purpose MIPS value is calculated using peak operation for best instruction execution time (one state time). New instructions were added to increase the controller's math performance for digital signal processing applications. Also, instruction execution times are significantly reduced in comparison to the previous MCS 96 controllers. For example, a two-operand multiplication operation using direct addressing is reduced from sixteen to three state times. This reduction helps increase the performance.

The 80296SA has 512 bytes of register RAM and 2 Kbytes of internal RAM for storing code or data. The 80296SA has the same peripherals as the 8*x*C196NP and 8*x*C196NU: an event processor array (EPA) with 80-ns resolution, a pulse-width modulator (PWM) with a maximum 97.6 kHz frequency, and a serial port with a maximum 12.5-Mbaud synchronous baud rate and a maximum 3.1-Mbaud asynchronous baud rate. However, its interrupt structure differs from that of the 8*x*C196NP and 8*x*C196NU, and it has no peripheral transaction server (PTS).

Additionally, like the 8*x*C196NU, the 80296SA includes phase-locked loop circuitry, which allows an external clock to drive the microcontroller at one-half or one-quarter the maximum internal clock frequency. The system is designed for using lower-frequency external clocks or oscillators, while maintaining the maximum internal operating frequency. The 80296SA also has an enhanced chip-select unit, interrupt controller, and timers. Finally, its windowing capability is enhanced to include the windowing of selected external memory locations for direct addressing.

This application note describes these enhancements and outlines differences to help migrate 8*x*C196NP or 8*x*C196NU designs to the 80296SA. Consult the *80296SA Microcontroller User's Manual* (order number 272803), the *80296SA Commercial CHMOS 16-bit Microcontroller* datasheet (order number 272748) for specifications, and the *80296SA Specification Update* (order number 272908) for in-depth descriptions and specifications.

2.0 ARCHITECTURAL DIFFERENCES

Table 1 lists the main features of the 8*x*C196NP, 8*x*C196NU, and 80296SA for comparison. The major differences of the 80296SA from the 8*x*C196NU are the following:

- pipelined architecture — a four-stage instruction pipeline with a 10-byte (rather than 8-byte) prefetch queue
- memory — no ROM, 512 bytes of register RAM, and 2 Kbytes of code/data RAM; an additional window-selection register (WSR1), plus the ability to window the code/data RAM or a section of external memory
- instruction set — new and enhanced instructions for digital signal processing; no instructions for PTS control; indirect and indexed PUSH and POP operations relative to the stack pointer work differently
- digital signal processing features — 40-bit hardware accumulator, barrel shifter, index registers, and new instructions
- bus timings — the ability to add up to 15 wait states to external bus cycles, different timing requirements for code and data fetches, a shorter address hold time in multiplexed mode, and automatic deferred bus cycles in demultiplexed mode

- addressing ability — 6 Mbytes of linear address space, achieved through chip-select enhancements
- interrupt structure — compatible with 8*x*C196NP and 8*x*C196NU, or programmable priorities and relocatable interrupt vector table; EXTINT*x* inputs can be programmed as either edge-triggered or level-sensitive inputs; no peripheral transaction server (PTS)
- peripheral enhancements — the serial port can handle mode 0 receptions at the highest possible baud rate; the event processor array's timers overflow on different boundaries when operating independently (not cascaded)
- power-saving features — the ability to disable the pulse-width modulator (PWM) and serial I/O (SIO) port

Table 1. Feature Comparisons

Feature	8*x*C196NP	8*x*C196NU	80296SA
Memory			
ROM	0 or 4 Kbytes	0 or 48 Kbytes	—
Register RAM	1 Kbyte	1 Kbyte	512 bytes
Code/data RAM	—	—	2 Kbytes
Address space	1 Mbyte	1 Mbyte	6 Mbytes
Frequency	25 MHz	40 MHz, 50 MHz	40 MHz, 50 MHz
Clock circuitry	Standard MCS® 96 clock circuitry	Phase-locked loop and clock multiplier circuitry (1X, 2X, 4X external clock)	Phase-locked loop and clock multiplier circuitry (1X, 2X, 4X external clock)
Power-saving features	Idle and powerdown modes	Idle, powerdown, and standby modes	Idle, powerdown, and standby modes, plus ability to disable PWM and SIO peripherals
Digital signal processing features	—	32-bit hardware accumulator with status register, modified multiply instruction to handle accumulator	40-bit hardware accumulator with status register, new instructions, index registers, hardware accumulator, and 32-bit barrel shifter
Interrupt structure	Standard MCS 96 interrupt controller and peripheral transaction server (PTS)	Standard MCS 96 interrupt controller and peripheral transaction server (PTS)	Compatible MCS 96 interrupt structure, or programmable interrupt priority and relocatable interrupt vector table. **No** peripheral transaction server (PTS)
Chip-select pins	6	6	6
External interrupt (EXTINT*x*) pins	4	4	4
Event processor array (EPA) pins	4	4	4
Pulse-width modulator (PWM) pins	3	3	3
Serial I/O (SIO) port	1 synchronous mode, 3 asynchronous modes	1 synchronous mode, 3 asynchronous modes	1 synchronous mode, 3 asynchronous modes
Packages	100-pin QFP and SQFP	100-pin QFP and SQFP	100-pin QFP

2.1 Pipelined Architecture

The 80296SA core implements a pipelined architecture, while maintaining binary code compatibility with other members of Intel's MCS 96 controller family. The pipeline has four stages: fetch, decode, read-execute, and execute-write. This design achieves a faster throughput of instructions than was possible with previous MCS 96 controllers. Figure 1 illustrates the four-stage pipeline.

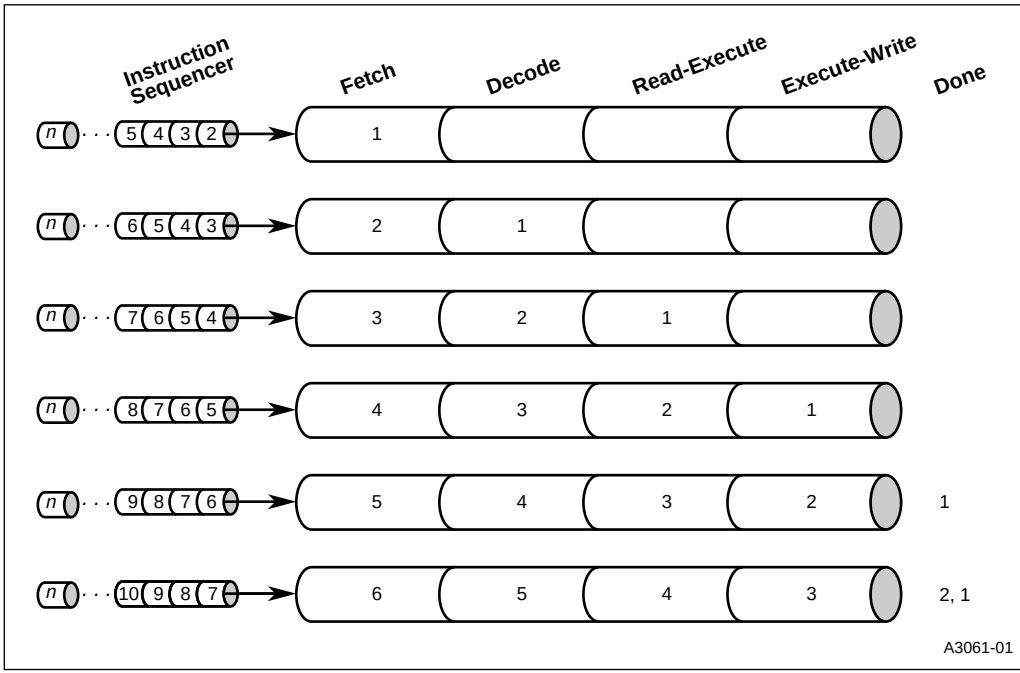

Figure 1. Pipelined Architecture

In a pipelined architecture, a different instruction is in each of the four pipeline stages. Instructions move sequentially through the pipeline stages. As one instruction moves from the fetch to the decode stage, the next instruction moves into the fetch stage. Similarly, the previous two instructions are in the read-execute and execute-write stages. In summary, four instructions can be in the pipeline at any given time, one in each of the four stages.

2.2 External Peripheral Interface Considerations

In earlier MCS 96 microcontrollers, the CPU completes execution of the current instruction before fetching the next instruction. If the write to an external device requires one or more wait states, the CPU of earlier MCS 96 microcontrollers stalls instruction execution until the write cycle completes.

However, because of its four-stage pipelined architecture, the 80296SA CPU does not stall instruction execution. If the write to an external device requires one or more wait states, the CPU continues executing instructions in the other three stages of the pipeline. Consider the case where a write to an external device occurs in the fourth pipeline stage (execute-write) and a subsequent read of an I/O port register occurs in the third pipeline stage (read-execute). The previous write instruction to an external device may not have completed while the following read instruction executes in the third pipeline stage. This may affect the system design if an external device returns a status signal to one of the microcontroller's ports based upon the write.

Figure 2 illustrates the 80296SA in a system with an ASIC (the ASIC could be any external peripheral). The figure shows the bus interface between the 80296SA and the ASIC. The dotted line shows the dependency of the bus information to the ASIC's output. For example, if the ASIC receives 0D0h from the controller, it may set port 1.x. This output signal is then fed to the controller's port 1. Even though the output is fed to port 1 in this example, the specific port is not important in this type of operation. The output could be fed to any available I/O port.

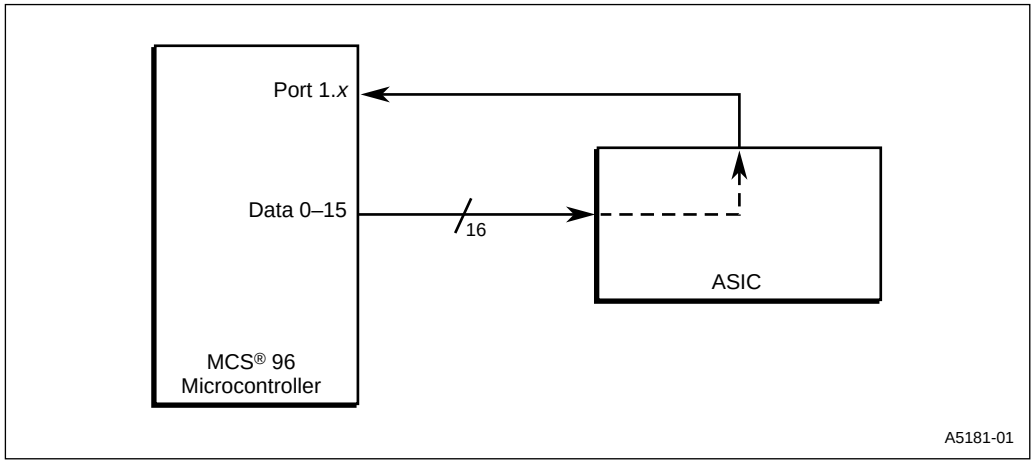

Figure 2. Interface Between 80296SA and an External Device

The following code sequence illustrates the write to the ASIC followed by a read of port 1:

```
ST    REG1, ASIC[0]   ; ASIC is the external device address
LDB   REG2, P1_PIN    ; Port 1.x receives a signal from the ASIC
                      ; reading P1_PIN will give info from ASIC
```

When the CPU executes the store (ST) instruction, it passes the write to the bus controller. At this point, the load byte (LDB) instruction is in the pipeline and executing. The CPU executes all possible pipeline stages of the load byte instruction (LDB) independent of the store (ST) instruction's stall in the fourth stage. Therefore, if a stall occurs, the load byte instruction might read P1_PIN before the store into the ASIC is complete. If this occurs, the ASIC has not received and processed the information from the controller in time to affect P1.*x* before port 1 is read. Therefore, the ASIC signal on port 1.x may send erroneous information. Figure 3 illustrates this timing sequence. Notice that the write is still in progress when the read of the internal register occurs.

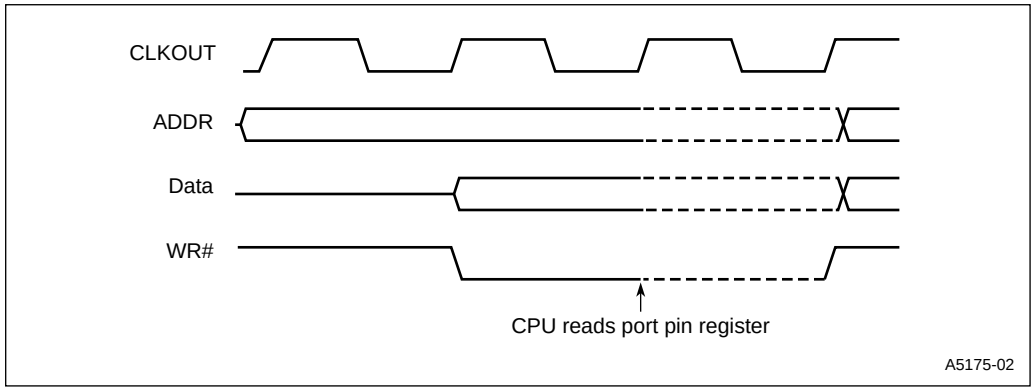

Figure 3. Timing of a Write Followed by an Internal Register Read

To avoid missing the external device's status signal when an external write is followed by an internal read, insert no-operation (NOP) instructions between the store and load instructions to cover the wait states required for the write.

2.3 Memory Differences

While the 8*x*C196NP and 8*x*C196NU have 1 Kbyte of register RAM (0–3FFh), the 80296SA has 512 bytes of register RAM (0–1FFh). The 80296SA will execute external memory fetches for locations 200–3FFh. Therefore, you must provide external memory at locations 200–3FFh if your application expects memory in these locations.

The 80296SA has added 2 Kbytes of code/data RAM. This RAM is mapped into a single address region (F800–FFFFh) in extended mode (1-Mbyte addressing mode), or it is mapped into two address regions (FFF800–FFFFFFh and F800–FFFFh) in nonextended mode (64-Kbyte addressing mode). This RAM may be used for time-critical code such as interrupt service routines, or time-critical data such as embedded digital signal processing data tables, the stack, or the interrupt vector table. System designers must determine appropriate allocation of this RAM for the system's time-critical use.

2.4 Windowing Differences

As on previous MCS 96 controllers, the 80296SA uses a window-selection register (WSR) to select a region of the upper register file or peripheral SFRs that can be addressed in the lower register file with direct addressing. WSR can select a 32-, 64-, or 128-byte region to map into registers located at E0–FFh, C0–FFh, or 80–FFh, respectively. Like the 8*x*C196NU, the 80296SA has a second window-selection register (WSR1). WSR1 can select a 32- or 64-byte memory region to map into registers located at 60–7Fh or 40–7Fh, respectively.

Additionally, in the 80296SA, the code/data RAM and selected external memory locations can be windowed into lower register RAM locations 40-7Fh. Windowing external memory locations has great implications on the performance. This feature allows you to perform a simple context switch of external memory and allow direct addressing of external memory or peripherals. Table 2 shows the windowable locations.

Table 2. Windowable Locations

WSR Window	Windowable Locations			
	Description	8*x*C196NP	8*x*C196NU	80296SA
WSR, 80–FFh (32, 64, or 128 bytes)	Upper register file Peripheral SFRs	0100–03FFh 1F00–1FDFh	0100–03FFh 1F00–1FFFh	0100–01FFh 1F00–1FFFh
WSR1, 60–7Fh (32 bytes or 64 bytes)	Upper register file Peripheral SFRs	— —	0100–03FFh 1F00–1FFFh	0100–01FFh 1F00–1FFFh
WSR1, 40–7Fh (64 bytes)	Code/data RAM External memory	— —	— —	F000–F7FFh F800–FFFFh

The PUSHA instruction does **not** push WSR1 onto the stack. If the interrupt routine or called subroutine modifies the WSR1 register, you must save the status of WSR1 by pushing and popping it from the stack or saving it to a temporary register.

3.0 INSTRUCTION SET DIFFERENCES

The instruction set was modified to improve handling of embedded digital signal processing routines. These instructions efficiently manipulate the 40-bit accumulator. The basic functions of the new accumulator instructions include the following:

- clearing the accumulator before execution (indicated by “Z” suffix)
- relocating the source within a data table (indicated by “R” suffix)
- signed or unsigned math (signed indicated by “S” prefix)
- saturating the accumulator value based on the result

The opcodes for the multiply-accumulate instructions are the same as those of the multiply instructions. The 80296SA differentiates the instructions as follows:

- If the destination operand is less than 10h, the 80296SA executes a multiply-accumulate instruction.
- If the destination operand is greater than or equal to 10h, the 80296SA executes a multiply instruction.

This convention works because addresses below 17h contain special-function registers (SFRs), which should never be the destination of a multiply instruction. The assembler translates the multiply-accumulate mnemonic into the appropriate opcode and destination operand.

Additionally, the repeat (RPT) instructions make handling repeated multiply-accumulate operations easier to implement in code. Also the return from interrupt (RETI) instruction was added to reduce interrupt latency. The opcode for the repeat instruction is the same opcode as that of the AND instructions. The microcontroller differentiates the instructions by the destination operand.

3.1 New and Modified Instructions

Table 3 lists the new and modified instructions on the 80296SA. See the *80296SA Microcontroller User’s Manual* for details on these instructions.

Table 3. New and Modified Instructions for the 80296SA

Instruction	Description
ADDC	Saturated addition to accumulator
SUBC	Saturated subtraction from accumulator
MAC	Unsigned multiply-accumulate
MACR	Unsigned multiply-accumulate with automatic data-move
MACRZ	Unsigned, zero accumulator, multiply-accumulate with automatic data-move
MACZ	Unsigned, zero accumulator, multiply-accumulate
SMAC	Signed multiply-accumulate
SMACR	Signed multiply-accumulate with automatic data-move
SMACRZ	Signed, zero accumulator, multiply-accumulate with auto-data-move
SMACZ	Signed, zero accumulator, multiply-accumulate
MSAC	Move saturated and shifted long word through barrel shifter
MVAC	Move shifted long word from accumulator
RPT	Repeat next instruction n times (n = repeat count)
RPTxxx	Repeat next instruction conditionally n times (n = repeat count; xxx = condition from conditional jump instruction)
RPTI	Interruptable repeat next instruction n times, n = repeat count operand
RPTIxxx	Interruptable repeat next instruction conditionally n times (n = repeat count; xxx = condition from conditional jump instruction)
RETI	Return from interrupt

3.2 Removed Peripheral Transaction Server Instructions

The 80296SA has no peripheral transaction server (PTS) interrupt structure; therefore, the DPTS and EPTS instructions were deleted from the instruction set.

3.3 Illegal Opcodes

In earlier MCS 96 microcontrollers, opcode EEh is reserved, but it does not execute an illegal opcode interrupt. Also, opcodes ECh and EDh are used to disable and enable the PTS, respectively. For the 80296SA, executing one of these three opcodes (ECh, EDh, or EEh) will execute an illegal opcode interrupt. As in previous controllers, executing opcode 10h will also generate an illegal opcode interrupt. Opcode E5h is an illegal opcode in earlier microcontrollers, but is the return-from-interrupt (RETI) opcode for the 80296SA. Table 4 summarizes the differences between these opcodes for the 80296SA and earlier MCS 96 controllers.

Table 4. Illegal Opcode Comparisons

Opcode	Previous MCS® 96 Controllers	80296SA Controller
10h	generates illegal opcode interrupt	generates illegal opcode interrupt
E5h	generates illegal opcode interrupt	executes return-from-interrupt (RETI) instruction
ECh	executes disable-PTS (DPTS) instruction	generates illegal opcode interrupt
EDh	executes enable-PTS (EPTS) instruction	generates illegal opcode interrupt
EEh	reserved; no illegal opcode interrupt	generates illegal opcode interrupt

3.4 Indirect and Indexed PUSH and POP Relative to the Stack Pointer

Indirect and indexed PUSH and POP operations relative to the stack pointer work differently on the 80296SA than on the 8*x*C196NP and 8*x*C196NU. The 8*x*C196NP and 8*x*C196NU microcontrollers calculate the address based on the value of the stack pointer **after** it is updated, but the 80296SA calculates the address based on the value of the stack pointer **before** it is updated.

4.0 PHASE-LOCKED LOOP AND CLOCK MULTIPLIER CIRCUITRY

The clock circuitry of the 80296SA and 8*x*C196NU includes a phase-locked loop and clock multiplier that enables the microcontroller to attain the maximum operating frequency of 50 MHz with an external clock source of either 50 MHz, 25 MHz, or 12.5 MHz (or an external crystal oscillator of 12.5 MHz or 25 MHz). The PLLEN1 and PLLEN2 pins control the clock multiplier, as shown in Table 5.

Table 5. Relationships Between Multiplier, Input Frequency, and Operating Frequency

Multiplier	PLLEN2:1 †	Maximum External Clock Frequency	Maximum External Oscillator Frequency	Internal Operating Frequency
1	00	50 MHz	25 MHz	50 MHz or 25 MHz
2	01	25 MHz	25 MHz	50 MHz
4	11	12.5 MHz	12.5 MHz	50 MHz

† PLLEN2:1 = 10 is a reserved combination that will cause the microcontroller to enter an unsupported test mode.

5.0 ACCUMULATOR AND INDEX REGISTERS

Several enhancements implemented with the 80296SA architecture are used in embedded digital signal processing. The enhancements increase the overall math performance over previous MCS 96 controllers. The following sections describe the accumulator and index registers. The sections also include application examples using the accumulator, index registers, and new instructions including the rotate feature for data movement.

5.1 Accumulator

The 40-bit accumulator is 32 bits with an overflow into another 8 bits. A control and status register (ACC_STAT) controls fractional and saturation modes and indicates the status of the accumulator's contents. The 8*x*C196NU has the same register for its 32-bit accumulator.

Because the accumulator is part of the core and not a special function register, only a limited number of operations are possible. The following instructions are valid for the lower 32 bits of the accumulator (ACC_00 and ACC_02):

- all eight MAC-related instructions
- LD or ST instructions on words starting at ACC_00 and ACC_02
- ADD or SUB instructions on the word starting at ACC_00
- ADDC or SUBC instructions on the word starting at ACC_02
- MVAC and MSAC instructions

- CMPL, SHLL, SHRAL, and NORML instructions

The following instructions are valid for the upper 8 bits of the accumulator (ACC_04):

- LDB or STB instructions on byte starting at ACC04 to a word-aligned boundary
- MAC, MACR, MACRZ, and MACZ instructions in conjunction with the lower 32 bites of the accumulator

5.2 Index Registers

The 80296SA has three pairs of index registers:

- index pointer registers, IDX0 and IDX1
- index reference registers, ICX0 and ICX1
- index control byte registers, ICB0 and ICB1

Use the index pointer registers, IDX0 and IDX1, as 24-bit pointers to any location within the 16-Mbyte address range. Use the index reference registers, ICX0 and ICX1, as destination and sources addresses to access the index pointer address locations. Use the index control byte registers, ICB0 and ICB1, to automatically increment or decrement the index pointer regsiters (by any value from 0 to 15) at the end of an instructions.

The following restriction apply to the index pointer registers, IDX0 and IDX1:

- IDX0 and IDX1 must be accessed with windowed direct addressing
- IDX0 must point to either a source 1 (SRC1) or a destination (DEST) address
- IDX1 must point to a source 2 (SRC2) address

To use these pointers, first load the index registers with the appropriate 24-bit starting address of the object being pointed to:

```
                                ; load IDX0 to point to 654321h
LDB     WSR, #7EH               ; select window 7Eh
LD      IDX0_7E, #4321h         ; load lower word of IDX0 with 4321h
LDB     IDX0_7E+2, #65h         ; load upper byte of IDX0 with 65h

                                ; load IDX1 to point to FEDCBAh
LD      IDX1_7E, #0DCBAh        ; load lower word of IDX1 with DBCAh
LDB     IDX1_7E+2, #0FEh        ; load upper byte of IDX1 with FEh
```

To enable the automatic incrementing or decrementing, program the control bytes:

```
LDB     WSR, #7EH               ; select window 7Eh
LD      ICB0, #3                ; auto-increment IDX0 by 3 bytes
LD      ICB1,#1Eh               ; auto-decrement IDX1 by 14 bytes
```

Finally, access the pointers via “dummy” locations, ICX0 and ICX1 (similar to *IDX0, *IDX1 from the C language). Note that the pointer registers are incremented or decremented only once per instruction, at the effective end of the instruction.

```
LD      ICX0, #20h          ; load #20 into location 654321h and increment IDX0 by 3
LD      R20, ICX1           ; load value in FEDCBAH into R20 and decrement ICX0 by 14
LD      ICX0, ICX1          ; load value of FEDCACh into location 654324H,
                            ; increment IDX0 by 3, and decrement IDX1 by 14
ADD     ICX0,ICX0,ICX1      ; add value in location 654327h to value in FEDC9Eh
                            ; and store result in location 654327h.
                            ; Increments IDX0 by 3 (even though it is used twice),
                            ; and decrements IDX1 by 14
```

5.3 Index Register Usage and Application Examples

The following examples illustrate allowable and prohibited usage of the index reference registers (ICX0 and ICX1). The examples assume that the index pointer registers (IDX0 and IDX1) are initialized and the index control bytes (ICB0 and ICB1) are properly configured.

5.3.1 ICX0, ICX1 Usage Allowed

```
LD      ICX0, ICX1              ; ICX0 on D1, ICX1 on S2
ST      R20, ICX0               ; using ICX0 only
ADD     ICX0, ICX1              ; ICX0 on D1/S1, ICX1 on S2
ADD     ICX0, ICX0, ICX1        ; equivalent to ADD ICX0,ICX1

LD      ICX0, [R20]+            ; using ICX0 only
LD      R20, ICX1               ; using ICX1 only
ADD     ICX0, R20, ICX1         ; ICX0 on DEST, ICX1 on SRC2
ADD     R20, ICX0, ICX1         ; ICX0 on SRC1, ICX1 on SRC2
```

5.3.2 ICX0, ICX1 Usage Not Allowed

```
LD      ICX1, ICX0              ; ICX0 and ICX1 on wrong address bus

ST      ICX1, ICX0              ; ICX0 could be on either S1 or S2

ADD     ICX0, ICX1, R20         ; ICX1 is on SRC1--wrong internal address bus
ADD     ICX0, ICX1, #0FFh       ; ICX1 is on SRC1--wrong internal address bus
ADD     ICX0, ICX1, [R20]+      ; ICX1 is on SRD1--wrong internal address bus

LD      ICX0, [ICX1]            ; wrong addressing mode--only register direct is allowed
LD      R20,[ICX1]+             ; wrong addressing mode--only register direct is allowed
```

5.3.3 Block Move Without BMOV

```
; Code segment to move 20 bytes of data from 333444h to 0FF1234h

LD      IDX0, #1234h            ; load destination pointer
LD      IDX0+2, #0FFh
LD      IDX1, #3444h            ; load source pointer
LD      IDX1+2, #33h

LD      ICB0, #1                ; increment destination table by 1 (for byte)
LD      ICB1, #1                ; increment source table by 1 (for byte)
RPT     #20                     ; move 20 bytes
LDB     ICX0, ICX1              ; move byte pointed to by ICX1 to wherever ICX0
                                ; points, and increment pointers each time
```

5.3.4 Table Multiply-accumulate

```
; Code segment to multiply-accumulate two 256-entry tables
; consisting of words (MAC mode assumed to be set up).
;
; Table 1 is at 112233h; table 2 is at 445566h and is to be rotated
; upward during the operation.
```

```
LD      IDX0, #2233h+#200       ; load SRC1 table pointer to top of table 1
LD      IDX0+2, #11h
LD      IDX1, #5566h+#200       ; load SRC2 table pointer to top of table 2
LD      IDX1+2, #44h
LD      ICB0, #12h              ; decrement by 2 for word on S1
LD      ICB1, #12h              ; decrement by 2 for word on S2

SMAC    ICX0, ICX1              ; MAC first entry at top of table
RPT     #99                     ; MAC with rotate rest of entries
SMACR    ICX0, ICX1             ; will rotate word at ICX1 to ICX1+2
                                ; for each MAC operation
```

6.0 INTERRUPTS

You can either program the interrupt controller on the 80296SA to perform like the previous MCS 96 controllers or program it with prioritized interrupts. After a system reset, the 80296SA defaults to the interrupt controller structure of previous MCS 96 microcontrollers — fixed priorities and vector table location (the vector table begins at FF2000h in the 1-Mbyte memory model or at 2000h in the 64-Kbyte memory model). The 80296SA allows you to program the priorities of maskable interrupts and to control the location of the interrupt vector table. Table 6 shows the default priority for each interrupt source, the allowable programmable priorities for each maskable interrupt source, and the default vector locations.

Table 6. Interrupt Sources, Vectors, and Priorities

Interrupt Source	Mnemonic	Name	Default Priority†	Programmable Priorities	Default Vector Location
Unimplemented Opcode	—	—	17††	17††	FF2012H
Software TRAP Instruction	—	—	16††	16††	FF2010H
Nonmaskable Interrupt	NMI	INT15	15††	15††	FF203EH
EXTINT3 Pin	EXTINT3	INT14	14	0–14	FF203CH
EXTINT2 Pin	EXTINT2	INT13	13	0–14	FF203AH
EPA2 & 3 Overruns	OVR2_3	INT12	12	0–14	FF2038H
EPA0 & 1 Overruns	OVR0_1	INT11	11	0–14	FF2036H
EPA Capture/Compare 3	EPA3	INT10	10	0–14	FF2034H
EPA Capture/Compare 2	EPA2	INT09	9	0–14	FF2032H
EPA Capture/Compare 1	EPA1	INT08	8	0–14	FF2030H
EPA Capture/Compare 0	EPA0	INT07	7	0–14	FF200EH
SIO Receive	RI	INT06	6	0–14	FF200CH
SIO Transmit	TI	INT05	5	0–14	FF200AH
EXTINT1 Pin	EXTINT1	INT04	4	0–14	FF2008H
EXTINT0 Pin	EXTINT0	INT03	3	0–14	FF2006H
Reserved	Reserved	INT02	2	0–14	FF2004H
Timer 2 Overflow	OVRTM2	INT01	1	0–14	FF2002H
Timer 1 Overflow	OVRTM1	INT00	0	0–14	FF2000H

† Upon reset, the 80296SA defaults to the 8*x*C196NU-compatible priority scheme. (The higher the number, the higher the priority.)

†† Fixed priority

6.1 Controlling the Interrupt Structure

The most-significant bit of the NMI_PEND register enables and disables the programmable-priority mode. Clear NMI_PEND.7 to use the default, fixed priorities; set NMI_PEND.7 to enable programmable priorities. Once NMI_PEND is initialized, program the INT_CON*x* registers to define the priority of each maskable interrupt. Each register maps specific interrupt sources to specific priorities and their corresponding vector addresses. With programmable priorities enabled, interrupt service routines must end with the RETI (return-from-interrupt) instruction rather than RET instruction.

6.2 Controlling the Interrupt Vector Table Location

You can locate the interrupt vector table anywhere in the address space on a 256-byte boundary. For faster execution of interrupt service routines, store the interrupt vector table in internal code RAM. To reassign the location, write the upper 16 bits of the interrupt vector table's base address to the VECT_ADDR register. When the CPU acknowledges an interrupt request, the interrupt controller generates an 8-bit jump address and adds it to the base address to generate a complete vector address. The 8-bit jump address represents the default vector location. The complete 24-bit vector address consists of VECT_ADDR (upper word) plus default vector location (lower byte).

6.3 Controlling EXTINT*x* Requests

The EXTINT_CON register allows you to individually select the action (rising edge, falling edge, high level, or low level) that causes an interrupt request on each EXTINT*x* input. The minimum level time is two states, and the minimum edge time is one state. (See "Power-saving Features" on page 14 for information on the treatment of EXTINT*x* inputs in power-saving modes.)

7.0 CHIP-SELECT UNIT

The chip select unit is similar to that of the 8*x*C196NP and the 8*x*C196NU, but it has the following additional features:

- It decodes all 24 bits of the internal address, allowing access to 6 Mbytes of address space.
- Its signals are prioritized (CS5# has the highest priority and CS0# has the lowest) to avoid potential bus contention.
- It can cause the bus controller to add from 0 to 15 wait states to external bus cycles.
- It can cause the bus controller to lengthen write operations, extending the data and address hold times by 1 state time.
- Its adjacent signals can be AND'ed together, so two chip-select signals can control the same memory chip with different bus configurations.

7.1 Address Decoding

The chip-select unit can decode the entire 24-bit internal address bus. The address compare (ADDRCOM*x*) and address mask (ADDRMSK*x)* registers of the 80296SA contain bits 23–8, while the registers of the 8*x*C196NP and 8*x*C196NU contain only bits 19–8. Therefore, the maximum addressable space using the chip select signals is 6 Mbytes: 1 Mbyte of address space for each of the six chip select signals. To illustrate this point, consider addresses FE2000h and EE2000h. These addresses are different internally, since all 24 address bits are available internally and decoded by the chip select unit. However, these addresses cannot be distinguished from each other externally, since only 20 external address pins are available. By decoding all 24 bits of the internal address, the chip-select unit can distinguish between these addresses.

7.2 Chip-select Signal Priority

The chip-select signals have been prioritized so that if two chip-select signals are active for the same address region, only one will be true. Chip select 0 has the lowest priority, and chip select 5 has the highest priority. Prioritizing the chip-select signals avoids contention for two chip selects trying to control the bus to different states. This feature is not available on the 8*x*C196NP and 8*x*C196NU devices.

7.3 Extending Write Cycles

The 80296SA has an additional bit (WH0) in the BUSCONx registers to enable you to extend a write cycle's hold time for address and data. Some memory devices require that the address be held after the rising edge of the write signal (WR#). To accommodate this requirement, the bus controller can hold the address and data an additional state after the rising edge of WR#. This feature can match the T_{WHAX} (A19:0, CS*x*# hold after WR# rising edge) timing of the microcontroller with the t_{WR} (address hold after write rising edge) timing of the memory device.

7.4 Controlling Wait States

The 8*x*C196NP and 8*x*C196NU chip-select units can cause the bus controller to add from 0 to 3 wait states to external bus cycles. The 80296SA can interface with slower memory devices because it can add from 0 to 15 wait states. Program the wait-state bits (WS3:0) in the BUSCON*x* register to select the desired number of wait states for each chip-select signal. Program chip configuration byte 0 (CCB0) to add 0, 5, 10, or 15 wait states to the bus cycle for the chip configuration byte 1 (CCB1) fetch.

After a reset, the following sequence ensures that the appropriate number of wait states are used:

1. The microcontroller fetches CCB0 with 15 wait states. This configuration allows the controller to interface with slow memory devices upon power-up or reset.
2. The 80296SA fetches CCB1 with the wait-state configuration you have previously programmed in CCB0.
3. At the beginning of your boot code, set up BUSCON5 (the default bus configuration).
4. Configure the rest of the system and program the remaining BUSCONx registers for the chip-select signals.

8.0 BUS CYCLE DIFFERENCES

The bus timings of the 80296SA allow the high-speed bus to interface with relatively slow code memory. (See the *80296SA Commercial CHMOS 16-bit Microcontroller* datasheet for specific timing information.) There are two main differences between the bus timings of the 80296SA and those of previous MCS 96 microcontrollers:

- a shorter address hold time after the falling edge of ALE (T_{LLAX}) for multiplexed bus cycles
- the microcontroller automatically invokes the deferred bus cycle mode for demultiplexed bus cycles

For a multiplexed bus cycle, the microcontroller exhibits a shorter address hold time (T_{LLAX}). This requires a fast latch. Because the address will not be valid as long as on previous MCS 96 microcontrollers, the delay on the address latch must be shorter.

For a demultiplexed bus cycle, the microcontroller **automatically** delays the WR# signal (and the next bus cycle) by one state (2t) in the first bus cycle following a chip-select change and in the first write cycle following a read cycle. This delay, called a deferred bus cycle, is designed to reduce bus contention when using slow memory devices. For the 8*x*C196NU, you can enable or disable deferred bus cycles by programming chip configuration byte 1 (CCB1).

9.0 PERIPHERAL ENHANCEMENTS

The serial I/O port, event processor array, and pulse-width modulator have been enhanced.

9.1 Serial I/O Port

Like the serial port of the 8*x*C196NU, the serial port of the 80296SA has a divide-by-two prescaler that is controlled by SP_CON.6. (SP_CON.6 was a reserved bit on the 8*x*C196NP and earlier MCS 96 microcontrollers.)

Unlike its predecessors, the serial port can handle mode 0 receptions at the highest possible baud rate. You can program the baud-rate register with 8001h (or 0001h) and receive the correct information from a mode 0 reception. (On previous MCS 96 microcontrollers, the minimum value for the baud-rate register is X002h, where X = 8 or 0.)

For additional power savings, the baud-rate counter is disabled after a power-up or reset. When writing configuration information to the serial port control register, clear SP_CON.7 to enable the baud-rate counter.

9.2 Event Processor Array

When the timers are cascaded, the timer1 overflow output is used as the timer 2 input. In this mode, the timers of the 80296SA overflow on the same boundaries as those of the 8*x*C196NU, 0001–0000h and FFFE–FFFFh. These boundaries compensate for internal delays to allow the cascading logic to operate.

When the timers of the 80296SA are operating independently (rather than cascaded), they overflow only on the 0000–FFFFh (or FFFF–0000h) boundary. If you implemented a workaround to check the overflow boundaries on an 80C196NU design, you will need to verify it for the 80296SA.

9.3 Pulse Width Modulator

For additional power savings, the duty-cycle counter is disabled after a power-up or reset. When writing configuration information to the PWM control register, clear CON_REG0.7 to enable the duty-cycle counter.

10.0 POWER-SAVING FEATURES

In addition to the idle and powerdown modes used for earlier MCS 96 microcontrollers, the 8*x*C196NU and the 80296SA have a standby mode.

10.1 Standby Mode

The standby current (I_{STDBY}) is less than 10% of the normal operating current (I_{CC}). Executing the IDLPD #3 instruction causes the microcontroller to enter standby mode. In standby mode, the CPU halts execution and internal logic freezes the internal CPU and peripheral clocks at logic state 0. The on-chip oscillator and phase-locked loop continue to operate. You must put the peripherals into a known state before entering standby mode, since the peripherals will stop functioning immediately after the standby mode command is executed. Allow the serial port to complete any transmission or reception that is in progress, then disable receptions before executing the IDLPD #3 instruction. While an exit from powerdown mode requires a delay to allow the phase-locked loop and the on-chip oscillator to stabilize, an exit from standby mode does not.

10.2 Disabling Powerdown and Standby Modes

CCB0.0 enables or disables entry into powerdown and standby modes. The microcontroller will not enter powerdown or standby mode if CCB0.0 is clear. This effectively avoids inadvertent entry into either powerdown or standby mode.

10.3 Additional Power-saving Options

For additional power conservation, the baud-rate and duty-cycle counters are disabled after a power-up or reset. If your design uses the serial port or the PWM, your initialization code must enable the counters by clearing the associated bits. SP_CON.7 controls the baud-rate counter, and CON_REG0.7 controls the duty-cycle counter.

11.0 SPECIAL-FUNCTION REGISTERS

The special-function registers of the 80296SA are essentially the same as those of the 8*x*C196NU, with some exceptions. Table 7 lists all CPU special-function registers of the80296SA; **bold** type indicates differences from the 8*x*C196NU. Table 8 lists only those peripheral SFRs that are unique to the 80296SA.

Table 7. CPU Special-Function Registers

Address	High (Odd) Byte	Low (Even) Byte
0016h	**ICX1 (H)**	**ICX1 (L)**
0014h	WSR1	WSR
0012h	INT_MASK1	INT_PEND1
0010h†	**ICX0 (H)**	**IXC0 (L)**
000Eh†	ACC_02 (H)	ACC_02 (L)
000Ch†	ACC_00 (H)	ACC_00 (L)
000Ah	ACC_STAT	Reserved
0008h	INT_PEND	INT_MASK
0006h	**Reserved**	**ACC_04**
0004h†	**RPT_CNT (H)**	**RPT_CNT (L)**
0002h	ONES_REG (H)	ONES_REG (L)
0000h	ZERO_REG (H)	ZERO_REG (L)

† Must be addressed as a word.

Table 8. Peripheral SFRs That Are Unique to the 80296SA

Address	High (Odd) Byte	Low (Even) Byte
1FF0h†	VECT_ADDR (H)	VECT_ADDR (L)
1FEEh†	INT_CON3 (H)	INT_CON3 (L)
1FECh†	INT_CON2 (H)	INT_CON2 (L)
1FEAh†	INT_CON1 (H)	INT_CON1 (L)
1FE8h†	INT_CON0 (H)	INT_CON0 (L)
…	…	…
1FCCh	Reserved	EXTINT_CON
1FCAh†	IN_PROG1 (H)	IN_PROG1 (L)
1FC8h	NMI_PEND	INT_PROG0
1FC6h	ICB1	IDX1 (H)††
1FC4h	IDX1 (M)††	IDX1 (L)††
1FC2h	ICB0	IDX0 (H)††
1FC0h	IDX0 (M)††	IDX0 (L)††

† Must be addressed as a word.

†† These 24-bit registers must be accessed with windowed direct addressing. Use a word instruction to access the lower word and a byte instruction to access the upper byte.