AP-716

APPLICATION NOTE

80960Cx/80960Jx/80960Hx Architectural Comparison

Tom Johnson
SPG 80960 Systems Engineer

Intel Corporation
Mail Stop CH6-311
5000 W. Chandler Blvd.
Chandler, Arizona 85226

January 29, 1996

Order Number: 272694-001

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

80960Cx/80960Jx/80960Hx ARCHITECTURAL COMPARISON

FIGURES

TABLES

1.0 Introduction

This document describes three implementations of the i960® architecture: the 80960Cx, 80960Jx, and 80960Hx microprocessors.

During the 80960Jx and 80960Hx definition process, every attempt was made to maintain backward compatibility with the 80960Cx processor. To a large extent, this compatibility is maintained; most application code will run on each processor without modification. Due to enhancements to the bus interface, instruction set, and other refinements, complete compatibility was not maintained. This document discusses in detail those differences between the 80960Cx, 80960Jx, and 80960Hx microprocessors. Suggestions are given for mitigating their impact on future designs.

This document also provides hints for developing applications which the 80960Cx processor can use now, and can be 80960Hx-ready in the future. The reader should note that the 80960Hx processor is **not drop-in compatible** with the 80960Cx processor. However, system hardware can be designed such that it **will accept both the 80960Cx and 80960Hx processors in the same socket**.

This document contains condensed information from the *i960® Cx Microprocessor User's Manual* (Intel Literature Order No. 270710), *Application Note 506, Designing for 80960Cx and 80960Hx Compatibility* (Intel Literature Order No. 272556), the *i960® Jx Microprocessor User's Manual* (Intel Literature Order No. 272483), and the *i960® Hx Microprocessor User's Manual*, (Intel Literature Order No. 272484). The reader should refer to these documents for more detailed information.

1.1 i960® Processor Family

Table 1 summarizes the features of the 80960Cx, 80960Jx, and 80960Hx microprocessor families. Subsections which follow further describe each processor's features.

Table 1. Feature Summary of the 80960Cx, 80960Jx, and 80960Hx Processors (Sheet 1 of 2)

Feature	Processor		
	80960Cx	80960Jx	80960Hx
Core	Superscalar (maximum 3 inst/clk)	Scalar, clock doubled	Superscalar (max 3 inst/clk), clock doubled, clock tripled
External Bus	32-bit demultiplexed address and data	32-bit multiplexed address/data	32-bit demultiplexed address and data, parity on data
Instruction Cache	CA: 1 Kbyte, 2-way CF: 4 Kbytes, 2-way	4 Kbytes, 2-way	16 Kbytes, 4-way
Data Cache	CA: None CF: 1 Kbyte, direct map, write-through	2 Kbytes, direct map, write-through	8 Kbytes, 4-way, write-through
Data RAM	1 Kbyte, mapped from 000H to 3FFH	1 Kbyte, mapped from 0000H to 3FFH	2 Kbytes, mapped from 000H to 7FFH
Register Cache	5 frames, programmable to 15 frames (more than 5 uses Data RAM)	8 frames	5 frames, programmable to 15 frames (more than 5 uses Data RAM)
Memory-mapped Registers	No	Yes	Yes
Direct Memory Access (DMA) Controller	Yes	No	No

Table 1. Feature Summary of the 80960Cx, 80960Jx, and 80960Hx Processors (Sheet 2 of 2)

Feature	Processor		
	80960Cx	80960Jx	80960Hx
Interrupt Controller	Yes	Yes	Yes
Guarded Memory Unit	No	No	Yes
Timers	None	Two	Two
Power Supply	5V	5V or 3.3V	3.3V, 5V tolerant
JTAG	No	Yes	Yes

1.1.1 The 80960Cx Processor Family

The 80960Cx family of processors are superscalar implementations of the i960 microprocessor architecture, and feature demultiplexed, 32-bit address and data buses. The two members of this family, the CA and CF, can execute up to three instructions per clock cycle. Due to the high degree of parallelism and on-chip caches, these processors are well suited for high performance applications such as networking and high-end imaging.

The 80960CA includes a 1 Kbyte, two-way set associative instruction cache and 1 Kbyte of on-chip data RAM. Data RAM is located in the processor's address space from locations 0000.0000H to 0000.03FFH. The CF processor enhances the CA feature set by increasing the instruction cache size to 4 Kbytes, and adding a 1 Kbyte direct-mapped write-through data cache. As with the CA, the CF contains 1 Kbyte of on-chip data RAM located from 0000.0000H to 0000.03FFH.

The upper 16 Mbytes of the address space (FF00.0000H through FFFF.FFFFH) are reserved for implementation-specific functions. In general, an application must not access this space unless specifically required by the implementation. In the case of the CA and CF processors, the application must locate the Initialization Boot Record (IBR) at address FFFF.FF00H.

Integrated peripherals include the Interrupt Controller and Direct Memory Access (DMA) Controller. The DMA peripheral is not available on the 80960Jx and 80960Hx processors.

1.1.2 The 80960Jx Processor Family

The 80960Jx family of microprocessors represent the next generation of high-performance, low-cost processors based on the i960 architecture. 80960Jx processors feature a 32-bit multiplexed address and data bus; the core issues one instruction per clock cycle. The 80960Jx processor family supports 5V and 3.3V operation.

These processors provide a maximum 4 Kbyte, two-way set associative instruction cache, and a maximum 2 Kbyte, direct-mapped write-through data cache. The 80960Jx features 1 Kbyte of on-chip data RAM located from address 0000.0000H to 0000.03FFH.

The upper 16 Mbytes of the address space (FF00.0000H through FFFF.FFFFH) are reserved for implementation-specific functions. In general, an application must not access this space unless specifically required by the implementation. Memory-mapped control registers are located within this address space:

- user/supervisor accessible control registers are located from address FF00.0000H to FF00.7FFFH
- supervisor only accessible control registers from address FF00.8000H to FFFF.FFFFH

The application must locate the Initialization Boot Record (IBR) at address FEFF.FF30H.

Integrated peripherals include a 80960Cx-compatible Interrupt Controller, and two 80960Hx-compatible 32-bit timers. **The 80960Jx does not implement a DMA peripheral.**

1.1.3 The 80960Hx Processor Family

The 80960Hx family of microprocessors represent the next generation of very high-performance superscalar processors based on the i960 architecture. 80960Hx processors feature 32-bit demultiplexed address and data buses; the core may issue up to three instructions per clock cycle. The 80960Hx processor family is fabricated on a 3.3V process. All internal logic and memories operate at 3.3V; however, the pad-ring may be biased at 5V, meaning that the 80960Hx can operate in a 5V system. To do so, the system must provide a low-current 5V supply to bias the I/O buffers. In any case, the system must provide a 3.3V supply.

These processors provide a 16 Kbyte, four-way set associative instruction cache, and 8 Kbyte, four-way set-associative write-through data cache. The 80960Hx includes 2 Kbytes of on-chip data RAM located from address 0000.0000H to 0000.07FFH.

The upper 16 Mbytes of the address space (FF00.0000H through FFFF.FFFFH) are reserved for implementation-specific functions. In general, an application must not access this space unless specifically required by the implementation. Memory-mapped control registers are located within this address space:

- user/supervisor accessible control registers are located from address FF00.0000H to FF00.7FFFH
- supervisor only accessible control registers from address FF00.8000H to FFFF.FFFFH

The application must locate the Initialization Boot Record (IBR) at address FEFF.FF30H.

Integrated peripherals include a 80960Cx-compatible Interrupt Controller, two 80960Jx-compatible 32-bit timers, and the Guarded Memory Unit (GMU). **The 80960Hx does not implement a DMA peripheral.**

2.0 Initialization

This section outlines differences in initialization data structures and initialization procedures of the 80960Cx, 80960Jx, and 80960Hx. Only differences between these processor families are highlighted.

2.1 Initialization Data Structures

This section describes the differences between the necessary initialization data structures. Subsections include Section 2.1.1, Initialization Boot Record (IBR), Section 2.1.2, Processor Control Block (PRCB), Section 2.1.3, Fault Table, Section 2.1.4, Control Table, Section 2.1.5, Arithmetic Control Register Initial Image, and the rest.

2.1.1 Initialization Boot Record (IBR)

The 80960Cx, 80960Jx, and 80960Hx share similar initialization mechanisms. All processors fetch an Initialization Boot Record (IBR) from a fixed location in memory. The IBR contains the address of the first instruction to fetch and the pointer to the Processor Control Block (PRCB), which is the same format for all processors. The IBR structure is identical for the 80960Cx, 80960Jx, and 80960Hx, and is given in Figure 1. However, it is important to note that the interpretation of the **InitBusConx** bytes varies across implementations.

InitBusConx initializes:

- MCON0 on the 80960Cx
- PMCON14:15 on the 80960Jx
- PMCON15 on the 80960Hx

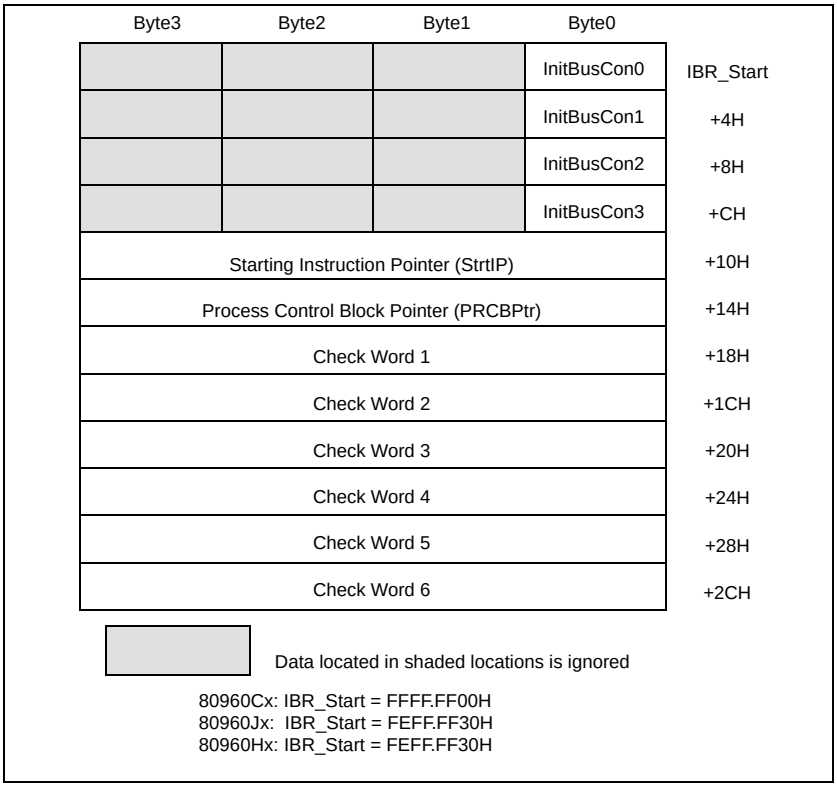

Figure 1. Initialization Boot Record (IBR) Structure

As indicated in Figure 1, the IBR contains the address of the first instruction to be fetched (**StrtIPx**), and the pointer to the PRCB (**PRCBPtrx**). The IBR also contains the initial bus configuration (**InitBusConx**), and check words for the bus confidence test. The algorithm used for the bus confidence test is identical for all processors.

The 80960Cx locates the IBR structure at address **FFFF.FF00H**; the 80960Jx and 80960Hx locate the IBR structure at address **FEFF.FF30H**. This was done so that a single boot ROM could hold two separate IBRs; one IBR for the 80960Cx, and one IBR for the 80960Jx or 80960Hx. This scheme works because of the way accesses to system memory are typically performed. Separate IBRs are necessary because of differences in initial register images, interpretation of the **InitBusConx** bytes, and control table structures. To clarify, refer to Figure 2.

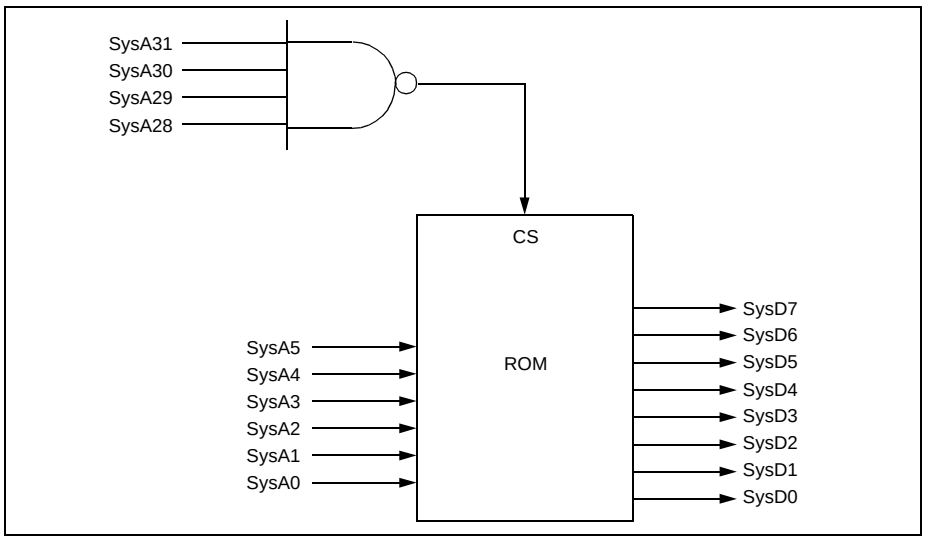

Figure 2. Simplified Typical Memory Decoding Logic

As can be seen from the figure above, system designers typically do not decode all high-order address pins when accessing memory. This allows for addresses to be aliased; logically different addresses will access the same physical memory.

Assume that the ROM in the figure above contains 64 Kbytes. In this case, when the 80960Cx accesses its IBR at FFFF.FF00H, address FF00H in the ROM is physically accessed. Likewise, when the 80960Jx or 80960Hx access the IBR at FEFF.FF30H, address FF30H in the ROM is physically accessed. Hence, both IBRs may be conveniently contained in the same ROM device. In the figure above, A24 can be ignored with the same results.

2.1.2 Processor Control Block (PRCB)

PRCB structure, illustrated in Figure 3, is identical for the 80960Cx, 80960Jx, and 80960Hx. Even though the PRCB structures are identical, the application designed to accommodate the 80960Cx and 80960Hx can be designed to provide separate control structures for each. This allows the application to define which control structures are common to the 80960Cx and 80960Hx, and which are unique.

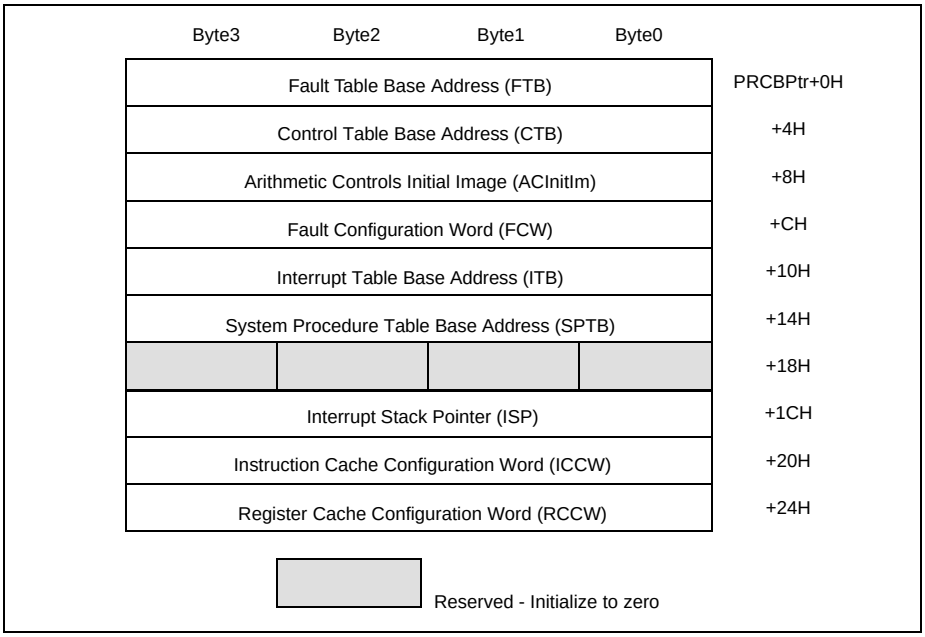

Figure 3. Processor Control Block (PRCB) Structure

As shown in Figure 1. Initialization Boot Record (IBR) Structure, the **PRCBPtr** is found in the IBR.

2.1.3 Fault Table

As indicated in Figure 3, the pointer to the fault table is found in the PRCB at offset 0H and is referred to as **FTB** (fault table base address). It may be initialized to any valid, non-reserved location in memory. The **FTB** points to the beginning of the fault table, which contains entries for the beginning address of each fault handler.

The fault tables for the 80960Cx, 80960Jx, and 80960Hx processors are identical, with the exception that the 80960Hx adds an entry for machine faults and an additional fault sub-type to support the Guarded Memory Unit. Machine faults are generated in response to parity errors. This is illustrated in Figure 4.

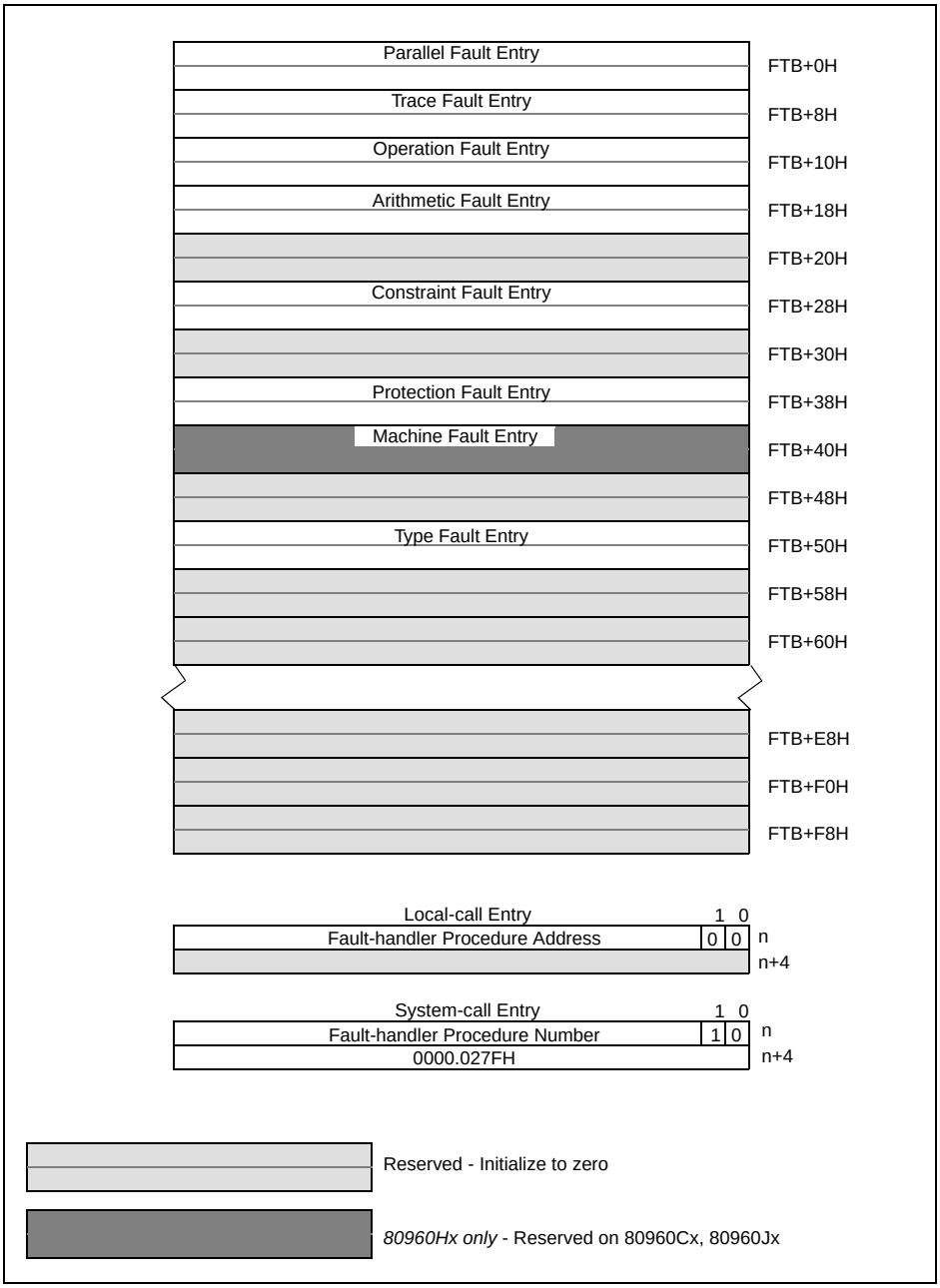

Figure 4. 80960Cx 80960Jx, and 80960Hx Fault Table and Fault Table Entries

The fault types and subtypes are summarized for the 80960Cx, 80960Jx, and 80960Hx processors. Each processor's users manual provides information on each fault type.

Table 2. 80960Cx, 80960Jx, and 80960Hx Fault Types and Subtypes

Fault Type		Fault Subtype		Fault Record (Hex)
Number	Name	Number/Bit Position	Name/Description	
00H	Parallel	02H through FFH	Indicates the number of faults that occur in parallel	XX00 XX02 XX00 XXFF
01H	Trace	Bit 0 Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7	Reserved Instruction Trace Branch Trace Call Trace Return Trace Prereturn Trace Supervisor Trace Breakpoint Trace	 XX01 XX02 XX01 XX04 XX01 XX08 XX01 XX10 XX01 XX20 XX01 XX40 XX01 XX80
02H	Operation	01H 02H 03H 04H	Invalid Opcode Unimplemented Unaligned Invalid Operand	XX02 XX01 XX02 XX02 XX02 XX03 XX02 XX04
03H	Arithmetic	01H 02H	Integer Overflow Arithmetic Zero-divide	XX03 XX01 XX03 XX02
04H	Reserved			
05H	Constraint	01H 02H	Constraint Range Privileged	XX05 XX01 XX05 XX02
06H	Reserved			
07H	Protection	Bit 1 Bit 5	Length Bad Access[1]	XX07 XX01 XX07 XX20
08H	Machine[2]	Bit 1	Parity Error	XX08 XX02
09H	Reserved			
0AH	Type	01H	Type Mismatch	XX0A XX01
0BH through 0FH	Reserved			

NOTES:

1. 80960Hx only. Added the Bad Access fault sub-type to support illegal memory accesses detected by the Guarded Memory Unit (GMU).
2. 80960Hx only. Added the Machine fault, with the Parity Error fault sub-type due to data parity supported on the 80960Hx data bus.

2.1.4 Control Table

As shown in Figure 3, the pointer to the control table is found in the PRCB at offset 4H and is referred to as **CTB** (control table base address). It may be initialized to any valid, non-reserved location in memory. **CTB** points to the beginning of the control table, which contains entries for the initial values of the memory configuration, breakpoint, interrupt control, and additional registers.

The control table for the 80960Cx, 80960Jx, and 80960Hx processors is shown in Table 3 Unique control tables are required for each processor, due to differences in the region and bus control registers. Refer to each processor's users manual for additional details

Table 3. Control Table for the 80960Cx, 80960Jx, and 80960Hx Processors (Sheet 1 of 3)

Offset	Description	Group
CTB + 00H	80960Cx: Instruction Breakpoint 0 (IPB0) 80960Jx: Reserved 80960Hx: Reserved	0H
CTB + 04H	80960Cx: Instruction Breakpoint 1 (IPB1) 80960Jx: Reserved 80960Hx: Reserved	
CTB + 08H	80960Cx: Data Address Breakpoint 0 (DAB0) 80960Jx: Reserved 80960Hx: Reserved	
CTB + 0CH	80960Cx: Data Address Breakpoint 1 (DAB1) 80960Jx: Reserved 80960Hx: Reserved	
CTB + 10H	80960Cx: Interrupt Map 0 (IMAP0) 80960Jx: Same 80960Hx: Same	1H
CTB + 14H	80960Cx: Interrupt Map 1 (IMAP1) 80960Jx: Same 80960Hx: Same	
CTB + 18H	80960Cx: Interrupt Map 2 (IMAP2) 80960Jx: Same 80960Hx: Same	
CTB + 1CH	80960Cx: Interrupt Control (ICON) 80960Jx: Same 80960Hx: Same	
CTB + 20H	80960Cx: Memory Region 0 Configuration (MCON0) 80960Jx: Physical Memory Region 0:1 Configuration (PMCON0:1) 80960Hx: Physical Memory Region 0 Configuration (PMCON0)	2H
CTB + 24H	80960Cx: Memory Region 1 Configuration (MCON1) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 1 Configuration (PMCON1)	
CTB + 28H	80960Cx: Memory Region 2 Configuration (MCON2) 80960Jx: Physical Memory Region 2:3 Configuration (PMCON2:3) 80960Hx: Physical Memory Region 2 Configuration (PMCON2)	
CTB + 2CH	80960Cx: Memory Region 3 Configuration (MCON3) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 3 Configuration (PMCON3)	

Table 3. Control Table for the 80960Cx, 80960Jx, and 80960Hx Processors (Sheet 2 of 3)

Offset	Description	Group
CTB + 30H	80960Cx: Memory Region 4 Configuration (MCON4) 80960Jx: Physical Memory Region 4:5 Configuration (PMCON4:5) 80960Hx: Physical Memory Region 4 Configuration (PMCON4)	3H
CTB + 34H	80960Cx: Memory Region 5 Configuration (MCON5) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 5 Configuration (PMCON5)	
CTB + 38H	80960Cx: Memory Region 6 Configuration (MCON6) 80960Jx: Physical Memory Region 6:7 Configuration (PMCON6:7) 80960Hx: Physical Memory Region 6 Configuration (PMCON6)	
CTB + 3CH	80960Cx: Memory Region 7 Configuration (MCON7) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 7 Configuration (PMCON7)	
CTB + 40H	80960Cx: Memory Region 8 Configuration (MCON8) 80960Jx: Physical Memory Region 8:9 Configuration (PMCON8:9) 80960Hx: Physical Memory Region 8 Configuration (PMCON8)	4H
CTB + 44H	80960Cx: Memory Region 9 Configuration (MCON9) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 9 Configuration (PMCON9)	
CTB + 48H	80960Cx: Memory Region 10 Configuration (MCON10) 80960Jx: Physical Memory Region 10:11 Configuration (PMCON10:11) 80960Hx: Physical Memory Region 10 Configuration (PMCON10)	
CTB + 4CH	80960Cx: Memory Region 11 Configuration (MCON11) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 11 Configuration (PMCON11)	
CTB + 50H	80960Cx: Memory Region 12 Configuration (MCON12) 80960Jx: Physical Memory Region 12:13 Configuration (PMCON12:13) 80960Hx: Physical Memory Region 12 Configuration (PMCON12)	5H
CTB + 54H	80960Cx: Memory Region 13 Configuration (MCON13) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 13 Configuration (PMCON13)	
CTB + 58H	80960Cx: Memory Region 14 Configuration (MCON14) 80960Jx: Physical Memory Region 14:15 Configuration (PMCON14:15) 80960Hx: Physical Memory Region 14 Configuration (PMCON14)	
CTB + 5CH	80960Cx: Memory Region 15 Configuration (MCON15) 80960Jx: Reserved (initialize to zero) 80960Hx: Physical Memory Region 15 Configuration (PMCON15)	

Table 3. Control Table for the 80960Cx, 80960Jx, and 80960Hx Processors (Sheet 3 of 3)

Offset	Description	Group
CTB + 60H	80960Cx: Reserved (initialize to zero) 80960Jx: Reserved (initialize to zero) 80960Hx: Reserved (initialize to zero)	6H
CTB + 64H	80960Cx: Breakpoint Control (BPCON) 80960Jx: Same 80960Hx: Same	
CTB + 68H	80960Cx: Trace Controls (TC) 80960Jx: Same 80960Hx: Same	
CTB + 6CH	80960Cx: Bus Configuration Control (BCON) 80960Jx: Same 80960Hx: Same	

NOTES:

1. CTB = Control Table Base address, found in the Processor Control Block (PRCB).
2. Differences exist between the processors in the format of the IMAPx and ICON registers.
3. Differences exist between the processors in the format of the BCON, MCON, and PMCON registers.

Note that MCON0, PMCON14:15, and PMCON15 are initialized from the IBR at reset.

2.1.5 Arithmetic Control Register Initial Image

The arithmetic control register (AC) is defined identically for the 80960Cx, 80960Jx, and 80960Hx processors. The AC initial image is programmed in the PRCB at offset 8H, and is shown as **ACInitIm** in Figure 3, Processor Control Block (PRCB) Structure (pg. 6).

2.1.6 Fault Configuration Word

The fault configuration word is defined identically for the 80960Cx, 80960Jx, and 80960Hx processors. The fault configuration word is programmed in the PRCB at offset CH, and is shown as **FCW** in Figure 3, Processor Control Block (PRCB) Structure (pg. 6).

2.1.7 Interrupt Table

Recall that the pointer to the interrupt table base address is found in the PRCB at offset 10H. The interrupt table base address pointer is identified as **ITB** (interrupt table base address), and may be initialized to any valid, non-reserved location in memory. **ITB** points to the beginning of the interrupt table, which contains entries for the starting addresses (interrupt vectors) of all interrupt service routines.

The formats of the interrupt tables for the 80960Cx, 80960Jx, and 80960Hx processors are identical. Interrupt vectors may be internally cached in on-chip data RAM in the same manner as well. The system may provide unique versions of the interrupt table for the 80960Cx and 80960Hx processors, so that the interrupt enable (**inten**) and disable (**intdis**) instructions may be used, which are not available on the 80960Cx processor.

2.1.8 System Procedure Table

As shown in Figure 3, Processor Control Block (PRCB) Structure (pg. 6), the pointer to the system procedure table is found in the PRCB at offset 14H. It is identified as **SPTB** (system procedure table base address), and may be initialized to any valid, non-reserved location in memory. **SPTB** points to the beginning of the system procedure table, which contains entries for the starting addresses system procedures. System procedures may be invoked by executing the call system (**calls**) instruction, or by the fault generator.

The system procedure table formats for all three processors are identical.

2.1.9 Interrupt Stack Pointer

The interrupt stack pointer is defined identically for all three processors; its initial image is programmed in the PRCB at offset 1CH, and is shown as **ISP** in Figure 3, Processor Control Block (PRCB) Structure (pg. 6).

2.1.10 Instruction Cache Configuration Word

The instruction cache configuration word (ICCW) is defined identically for all three processors. The ICCW initial image is programmed in the PRCB at offset 20H, and is shown as **ICCW** in Figure 3.

2.1.11 Register Cache Configuration Word

The register cache configuration word (RCCW) initial image is programmed in the PRCB at offset 24H, and is shown as **RCCW** in Figure 3, Processor Control Block (PRCB) Structure (pg. 6). The 80960Jx and 80960Hx processors add additional programmability in addition to that provided for by the 80960Cx processor, to allow local register frames in the on-chip register cache to be reserved for use by high-priority interrupts. This involves the implementation of bits in the RCCW which are reserved on the 80960Cx processor. However, since the 80960Cx ignores these bits, one value can be programmed in the PRCB which will work properly for all three processors. Refer to section 4.0 Register Cache/Stack Frames, for more information.

2.2 Register Values after Reset/Reinitialization

This section describes the values of the architectural state of the 80960Cx, 80960Jx, and 80960Hx processors after hardware reset and software reinitialization.

2.2.1 Common Reset/Reinitialized States

Table 4 shows the inital state values held in common by all three processors after hardware reset and software reinitialization.

Table 4. Common State of the i960® Processors after Reset/Reinitialization (Sheet 1 of 2)

Register	Value after Hardware Reset	Value after Software Reinitialization
g0	Device ID	Device ID
AC	AC initial image at **PRCBptr + 8H**	AC initial image at **PRCBPtr + 8H**
TC	TC initial image at **CTB + 68H**	TC initial image at **CTB + 68H**
FP (g15)	**ISP** (**PRCBPtr + 1CH**)	**ISP** (**PRCBPtr + 1CH**)
PFP (r0)	Undefined	Value before Software Reinitialization
SP (r1)	**ISP** + 40H ([**PRCBPtr** + **1CH**] + 40H)	**ISP** + 40H ([**PRCBPtr** + **1CH**] + 40H)
RIP (r2)	Undefined	Undefined
IPND	Undefined	Value before Software Reinitialization
IMSK	00H	00H
IMAP0	CTB + 10H	CTB + 10H
IMAP1	CTB + 14H	CTB + 14H
IMAP2	CTB + 18H	CTB + 18H
ICON	CTB + 1CH	CTB + 1CH
80960Cx: MCON0 80960Jx: PMCON0:1 80960Hx: PMCON0	CTB + 20H	CTB + 20H
80960Cx: MCON1 80960Hx: PMCON1	CTB + 24H	CTB + 24H
80960Cx: MCON2 80960Jx: PMCON2:3 80960Hx: PMCON2	CTB + 28H	CTB + 28H
80960Cx: MCON3 80960Hx: PMCON3	CTB + 2CH	CTB + 2CH
80960Cx: MCON4 80960Jx: PMCON4:5 80960Hx: PMCON4	CTB + 30H	CTB + 30H
80960Cx: MCON5 80960Hx: PMCON5	CTB + 34H	CTB + 34H
80960Cx: MCON6 80960Jx: PMCON6:7 80960Hx: PMCON6	CTB + 38H	CTB + 38H
80960Cx: MCON7 80960Hx: PMCON7	CTB + 3CH	CTB + 3CH
80960Cx: MCON8 80960Jx: PMCON8:9 80960Hx: PMCON8	CTB + 40H	CTB + 40H
80960Cx: MCON9 80960Hx: PMCON9	CTB + 44H	CTB + 44H
80960Cx: MCON10 80960Jx: PMCON10:11 80960Hx: PMCON10	CTB + 48H	CTB + 48H

Table 4. Common State of the i960® Processors after Reset/Reinitialization (Sheet 2 of 2)

Register	Value after Hardware Reset	Value after Software Reinitialization
80960Cx: MCON11 80960Hx: PMCON11	CTB + 4CH	CTB + 4CH
80960Cx: MCON12 80960Jx: PMCON12:13 80960Hx: PMCON12	CTB + 50H	CTB + 50H
80960Cx: MCON13 80960Hx: PMCON13	CTB + 54H	CTB + 54H
80960Cx: MCON14 80960Jx: PMCON14:15 80960Hx: PMCON14	CTB + 58H	CTB + 58H
80960Cx: MCON15 80960Hx: PMCON15	CTB + 5CH	CTB + 5CH
BCON	CTB + 6CH	CTB + 6CH

2.2.2 Differences in the Reset/Reinitialized State

Table 5 shows the differences in the state of the 80960Cx, 80960Jx, and 80960Hx processors after hardware reset and software reinitialization.

Table 5. Differences in State of the i960® Processors after Reset/Reinitialization (Sheet 1 of 2)

Register	Value after Hardware Reset	Value after Software Reinitialization	Processor		
			Cx	Jx	Hx
PC	C01F.2002H	C01F.2002H	X		
	001F.2002H	001F.2002H		X	X
DMAC (sf2)	00H	00H	X		
CCON (sf2)	TBD [1]	TBD [1]			X
BPCON	**CTB + 64H**	**CTB + 64H**	X		
	0000.0000H	0000.0000H		X	X
LMARx	Undefined	Value before Software Reinitialization		X	X
LMMRx	Bit 0 = 0; bits 1 - 31 = undefined	Bit 0 = 0; bits 1 - 31 = undefined [3]		X	X
DLMCON	Bit 0 = bit 7 of byte **InitBusCon**$_3$; bit 1 = 0; bits 2 - 31 = undefined	Bit 0 = value before warm reset; bit 1 = 0; bits 2 - 31 = undefined		X	X
TRRx	Undefined	Value before software reinitialization		X	X
TCRx	Undefined	Value before software reinitialization		X	X
TMRx	Bits 1 - 6 = 0; bits 0, 7 - 31 = undefined [2]	Bits 1 - 6 = 0; bits 0, 7 - 31 = undefined [2]		X	X

Table 5. Differences in State of the i960® Processors after Reset/Reinitialization (Sheet 2 of 2)

Register	Value after Hardware Reset	Value after Software Reinitialization	Processor		
			Cx	Jx	Hx
GCR (sf4)	Bits 0 - 7 = 0; bits 8 - 31 = undefined	Bits 0 - 7 = 0; bits 8 - 31 = undefined			X
MPAR0 thru MPAR1	Undefined	Value before Software Reinitialization			X
MPMR0 thru MPMR1	Undefined	Value before Software Reinitialization			X
MDUB0 thru MDUB5	Undefined	Value before Software Reinitialization			X
MDLB0 thru MDLB5	Undefined	Value before Software Reinitialization			X
IPB0	CTB + 00H	CTB + 00H	X		
	0000.0000H	0000.0000H		X	X
IPB1	CTB + 04H	CTB + 04H	X		
	0000.0000H	0000.0000H		X	X
DAB0	CTB + 08H	CTB + 08H	X		
	0000.0000H	0000.0000H		X	X
DAB1	CTB + 0CH	CTB + 0CH	X		
	0000.0000	0000.0000H		X	X
IPB2 thru IPB5	0000.0000H	Value before Software Reinitialization			X
DAB2 thru DAB5	0000.0000H	Value before Software Reinitialization			X
XBPCON	0000.0000H	Value before Software Reinitialization			X
DEVICEID	Device Identification	Device Identification		X	X

NOTES:

1. Bit 31 = 1, all others = 0
2. Bits 1-5 = 0, bits 0, 6-31 = undefined
3. LMMRx retains value (Jx only) otherwise, bit 0 = 0, bits 1-31 = undefined

2.3 Initial Bus Configuration

As indicated in Figure 1, Initialization Boot Record (IBR) Structure, The **InitBusConx** bytes are used for the initial configuration of the bus controller. These bytes configure the bus controller during the interim period from the beginning of initialization until the remainder of the configuration data structures are loaded from external memory to fully configure the bus controller. Each processor uses these bytes to initialize the appropriate memory control register. Since each processor has unique memory control register formats, these bytes are interpreted differently for each processor. For more information on memory control, refer to Section 2.3.4, External Bus Configuration.

2.3.1 80960Cx Processor

The first four words of the IBR are fetched from memory with the most relaxed bus parameters, specifically:

- Non-burst
- Non-pipelined
- Ready disabled
- 8-bit bus width
- Little endian byte order
- $N_{RAD} = 31$
- $N_{RDD} = 3$
- $N_{WAD} = 31$
- $N_{WDD} = 3$
- $N_{XDA} = 3$

The 80960Cx provides memory control with 16 Memory Control registers, named MCON15:0. The MCON registers define the physical attributes of the external memory system: wait state profiles, bus width, pipelining, bursting, $\overline{\text{READY}}$ enable, etc. MCON0 controls external accesses to addresses **0000.0000H** through **0FFF.FFFFH**, MCON1 controls accesses **1000.0000H** through **1FFF.FFFFH**, and so forth. In this manner, the MCON registers control 256 Mbyte regions of memory. The bus controller interface is discussed in further detail in Section 2.3.4.1, External Memory Configuration on the 80960Cx Processor.

As the first four words of the IBR are read from external memory, the low-order byte from each word is loaded into MCON0. The exact order in which these words are read is implementation dependent. This means that **InitBusCon0** is loaded into $MCON0_0$, **InitBusCon1** into $MCON0_1$, and so forth. The most significant byte of all MCONx registers is reserved, and must be programmed to zero (**InitBusCon3** must be zero). Once MCON0 is configured, all subsequent external bus accesses are controlled by this register until the remaining MCONx and Bus Control (BCON) registers are configured. After the MCONx registers are configured, they are marked valid and then control accesses to their respective memory regions.

Typically, the IBR is located in ROM memory; MCON0 is initially loaded with the appropriate parameters to access this memory.

2.3.2 80960Jx Processor

The first four words of the IBR are fetched from memory with the most relaxed bus parameters, specifically:

- 8-bit bus width

Since there is no internal wait state generator on the 80960Jx, the external memory system must return $\overline{\text{READY}}$ as appropriate.

The 80960Jx allows enhanced memory control by providing eight Physical Memory Control registers (PMCONx:x+1), two Logical Memory Control register pairs (LMCONx), and the Default Logical Memory Control register (DLMCON). The PMCONx:x+1 registers define the bus width of the external memory system. The LMCONx register pairs define the logical characteristics of the external memory system which, in the case of the 80960Jx, is simply cacheability. The DLMCON register defines the default logical characteristics of the external memory system for addresses not specifically covered by the LMCONx register pairs. This means that the DLMCON register controls the default cacheability characteristic of external memory, as well as the globally controlling byte order (big- or little-endian). Byte order on the 80960Jx cannot be controlled by LMCONx register pairs on a region-by-region basis; the memory space is uniformly big- or little-endian.

PMCON0:1 controls the bus width for external accesses to addresses **0000.0000H** through **1FFF.FFFFH**, PMCON2:3 controls accesses **2000.0000H** through **3FFF.FFFFH**, and so forth. In this manner, the PMCONx:x+1 registers control 512 Mbyte regions of memory. The LMCON registers may be programmed to control a variable range of addresses. The bus controller interface is discussed in further detail in Section 2.3.4.2, External Memory Configuration on the 80960Jx Processor.

As the first four words of the IBR are read from external memory, the low-order byte from each word is loaded into memory configuration register 14:15 (PMCON14:15). The exact order in which these words are read is implementation dependent. This means that **InitBusCon0** is loaded into PMCON14:15$_0$, **InitBusCon1** into PMCON14:15$_1$, and so forth. Once PMCON14:15 is configured, all subsequent external bus accesses are controlled by this register until the remaining PMCONx:x+1 and Bus Control (BCON) registers are initialized. After the PMCONx:x+1 registers are initialized, they are marked valid and then control accesses to their respective memory regions. Bit 7 of **InitBusCon3** is loaded into bit 0 of the DLMCON register; this bit controls data endianess for the entire 32-bit memory space.

Typically, the IBR is located in ROM memory, and PMCON14:15 is initially loaded with the appropriate parameters to access this memory. Note that PMCON14:15 is initialized first from the IBR, as opposed to PMCON0:1 (or as MCON0 is on the 80960Cx). This makes sense since the IBR is fixed in memory region 15; accesses to this memory region are normally controlled by PMCON14:15, not PMCON0:1. On the 80960Cx, system programmers were often required to reconfigure MCON0 (which controls memory region zero) if the memory profile for this region did not match that of memory region 15.

2.3.3 80960Hx Processor

The first four words of the IBR are fetched from memory with the most relaxed bus parameters:

- Non-burst
- Non-pipelined
- Ready disabled
- 8-bit bus width
- Little endian byte order
- $N_{RAD} = 31$
- $N_{RDD} = 3$
- $N_{WAD} = 31$
- $N_{WDD} = 3$
- $N_{XDA} = 15$

The possible number of N_{XDA} wait states is increased from three on the 80960Cx, to fifteen on the 80960Hx.

The 80960Hx allows enhanced memory control by providing:

- 16 Physical Memory Control registers (PMCONx)
- 15 Logical Memory Control register pairs (LMCONx)
- Default Logical Memory Control register (DLMCON)

The PMCONx registers define the physical attributes of the external memory system: wait state profiles, parity, bus width, pipelining, bursting, $\overline{\text{READY}}$ enable, etc. The LMCONx register pairs define the logical characteristics of the external memory system: data endianess, data cache independently invalidatable regions, and cacheability. The DLMCON register defines the default logical characteristics of the memory system for addresses not specifically covered by the LMCONx register pairs.

PMCON0 controls external accesses to addresses **0000.0000H** through **0FFF.FFFFH**, PMCON1 controls accesses **1000.0000H** through **1FFF.FFFFH**, and so forth. In this manner, the PMCONx registers control 256 Mbyte regions of memory. The LMCONx register pairs may be programmed to control a variable range of addresses. PMCON registers (80960Hx) and MCON registers (80960Cx) are roughly analogous in functionality, if the endianess and cacheability bits are ignored. The bus controller interface is discussed in further detail in Section 2.3.4.3, External Memory Configuration on the 80960Hx Processor.

As the first four words of the IBR are successively read from external memory, the low-order byte from each word of the IBR is successively loaded into each byte of Memory Configuration register 15 (PMCON15). This means that **InitBusCon0** gets loaded into PMCON15$_0$, **InitBusCon1** into PMCON15$_1$, and so forth. Once PMCON15 is configured, all subsequent external bus accesses are controlled by this register until the remaining PMCONx and Bus Control (BCON) registers are initialized. After all PMCONx registers are initialized, they are marked valid and then control accesses to their respective memory regions.

Typically, the IBR is located in ROM memory, and PMCON15 is initially loaded with the appropriate parameters to access this memory. Note that PMCON15 is initialized first from the IBR, as opposed to PMCON0 (or as MCON0 is on the 80960Cx). This makes sense since the IBR is fixed in memory region 15; accesses to this memory region are normally controlled by PMCON15, not PMCON0. On the 80960Cx, system programmers were often required to reconfigure MCON0 (which controls

memory region zero) if the memory profile for this region did not match that of memory region 15.

2.3.4 External Bus Configuration

This section describes differences in the external bus configuration of the 80960Cx, 80960Jx, and 80960Hx processors. Specifically, this section outlines the differences in programming the Memory Region Configuration (MCON), Physical Memory Region Configuration (PMCON), Logical Memory Template Address (LMAR), Logical Memory Template Mask (LMMR), Default Logical Memory Configuration (DLMCON), and Bus Control (BCON) registers.

2.3.4.1 External Memory Configuration on the 80960Cx Processor

The 80960Cx processor's MCON registers define the physical nature of the memory system (wait state profile, ready enable, burst, pipeline, bus width), and logical nature (byte order, cacheability - CF only). There are 16 MCON registers, one for each 256-Mbyte physical memory region. These registers are labelled MCON0 through MCON15. The format of the MCON register is shown in Figure 5.

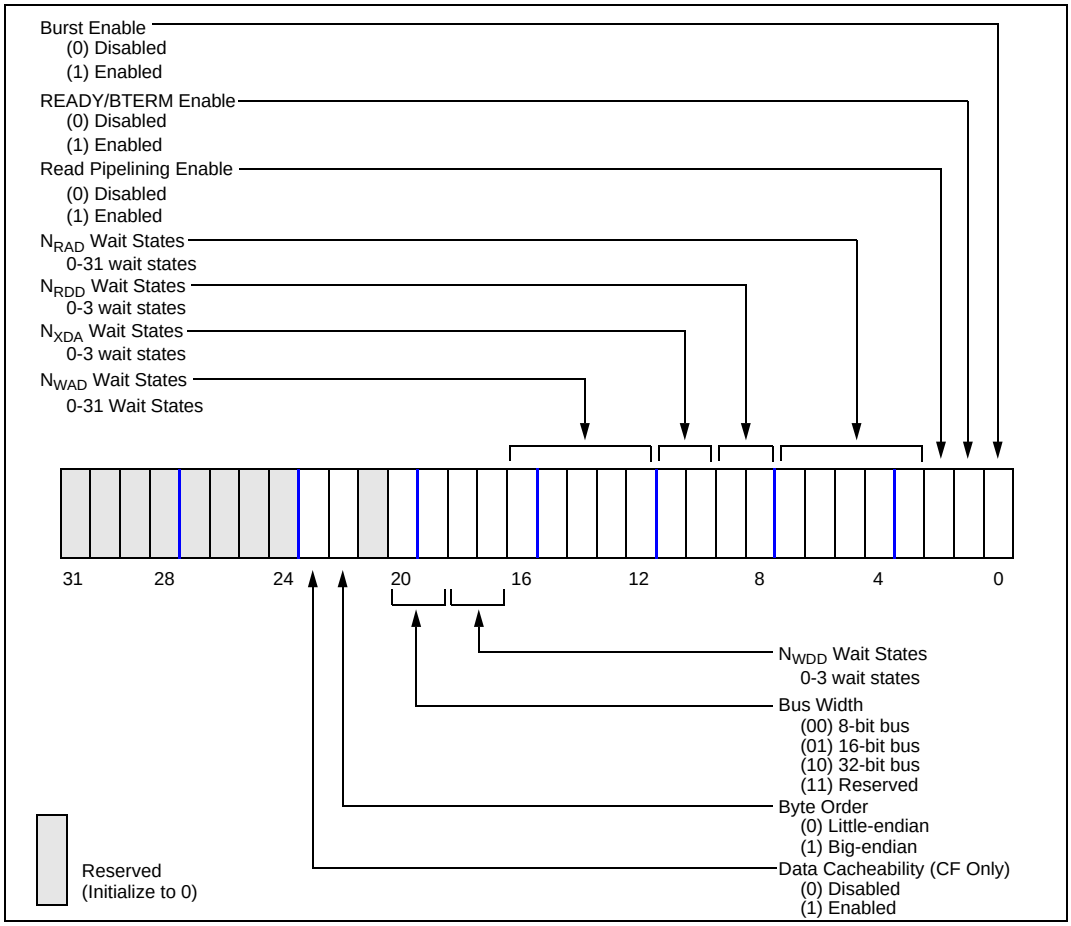

Figure 5. 80960Cx MCON Register Format

As shown in Figure 6, The Bus Control Register (BCON) mainly serves to validate the values stored in the MCON registers (region table). Immediately after a hardware reset, the region table is marked invalid in the BCON register. Whenever the region table is marked invalid in the BCON register (bit 0 = bit 0), the Bus Control Unit (BCU) uses the physical memory parameters found in MCON0 for accesses to all regions (Recall that MCON0 is initialized from data found in the IBR). Once the entire region table is loaded by reset microcode, the region table is marked valid in the BCON register (bit 0 = 1), and the BCU uses the appropriate user programmed values.

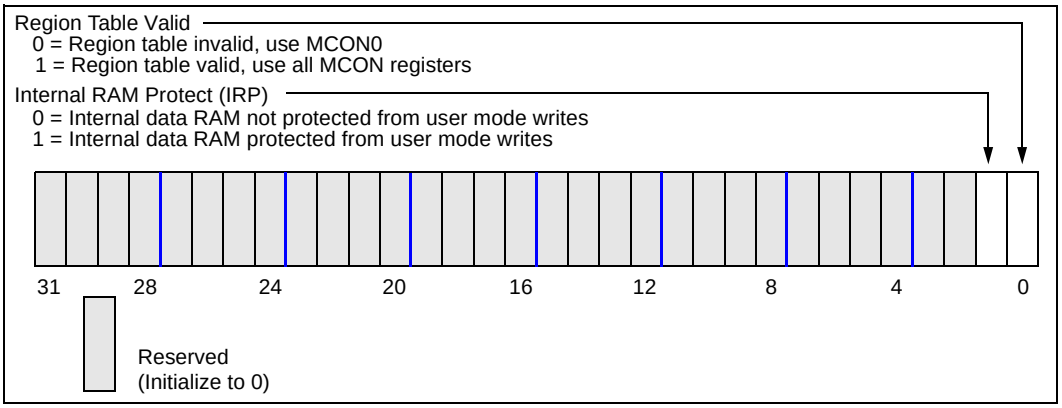

Figure 6. 80960Cx BCON Register

2.3.4.2 External Memory Configuration on the 80960Jx Processor

Control of physical and logical memory characteristics is divided among several registers on the 80960Jx:

- Physical Memory Region Control (PMCON)
- Logical Memory Template Address (LMAR)
- Logical Memory Template Mask (LMMR)
- Default Logical Memory Configuration (DLMCON) registers

There are eight PMCON registers, one for each 512 Mbyte physical memory region on the 80960Jx (PMCON0:1 through PMCON14:15). There are two pairs of LMAR/LMMR registers (LMAR0, LMAR1, LMMR0, LMMR1), and one DLMCON register.

PMCON registers control the bus width on the 80960Jx processor. This is shown in Figure 7.

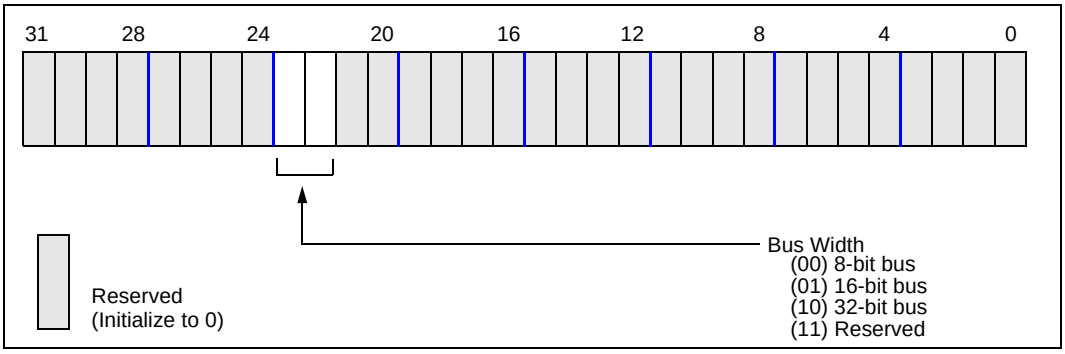

Figure 7. PMCON Register Format for the 80960Jx Processor

Logical memory control registers (LMARx and LMMRx) inform the processor logical attributes of external memory. On the 80960Jx, the only logical attribute controlled by logical memory register pairs is data cacheability. The low-order bit LMARx registers reflects the state byte-order control bit present in DLMCON. In the LMARx registers, this is a read only bit. In other words, the 80960Jx does not support multiple byte-order profiles by logical region; it implements only a homogeneous model. The entire memory map is either big-endian or little-endian.

Logical memory regions are created using the LMAR/LMMR register pair. Either:

- LMAR contains a starting address (4 Kbyte granularity), against which all external accesses are compared, or
- LMMR contains a mask, which indicates which address bits in LMAR are actually compared (compare under mask).

In this manner, one register pair can be used to create multiple logical regions, aliased on an arbitrary power-of-two boundary, by making some combination of upper memory bits “don’t care.” The format of the LMAR and LMMR registers is shown in Figure 8.

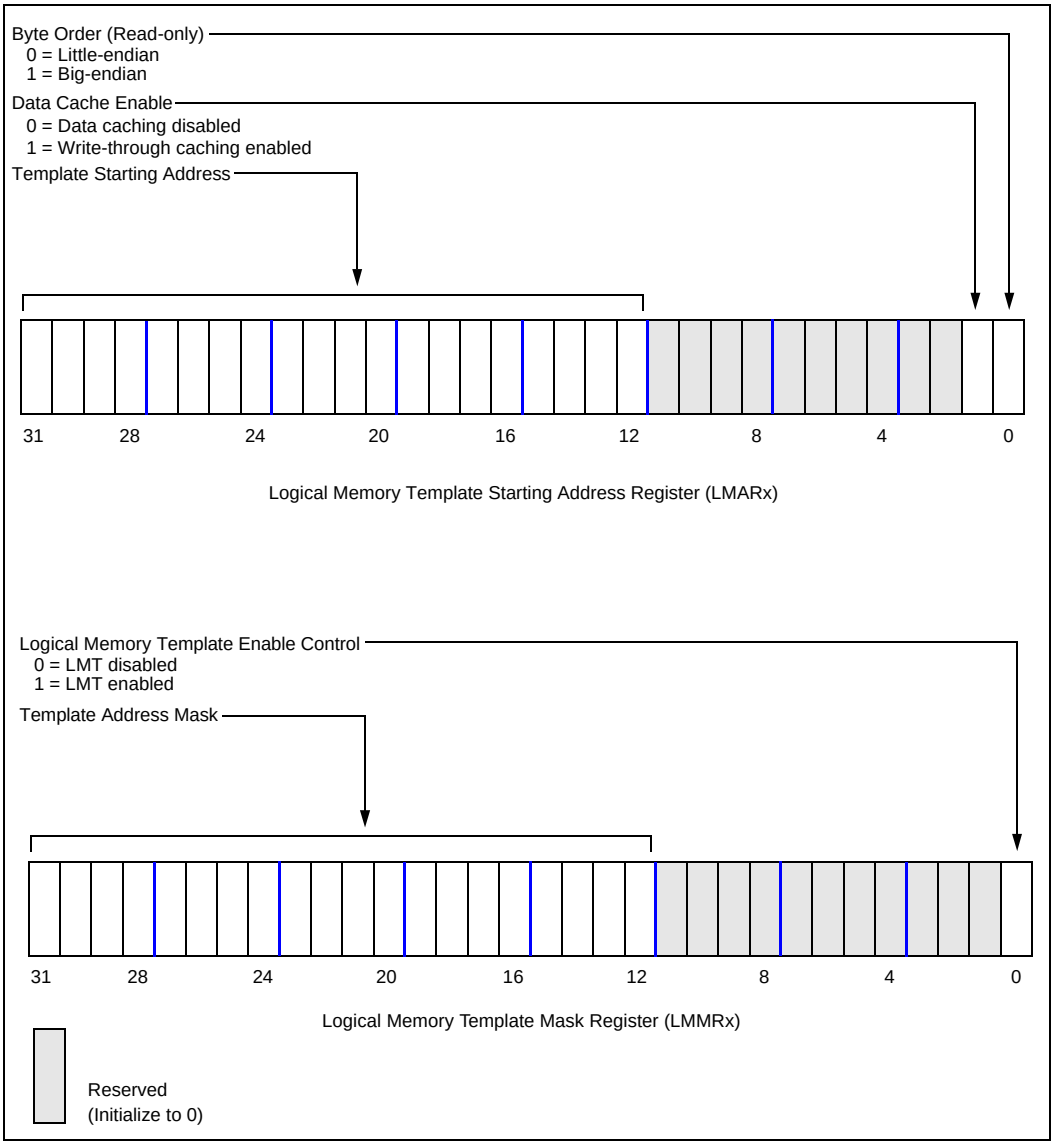

Figure 8. Format of LMARx and LMMRx Pairs on the 80960Jx Processor

The Default Logical Memory Configuration Register (DLMCON) provides default logical memory control for those accesses which do not fall within a region defined by the logical memory control register pairs. Logical attributes controlled by the DLMCON include data caching enable and byte ordering. On the 80960Jx, the byte order programmed in the DLMCON register controls byte ordering for the entire 32-bit memory space. The format of the DLMCON is illustrated in Figure 9.

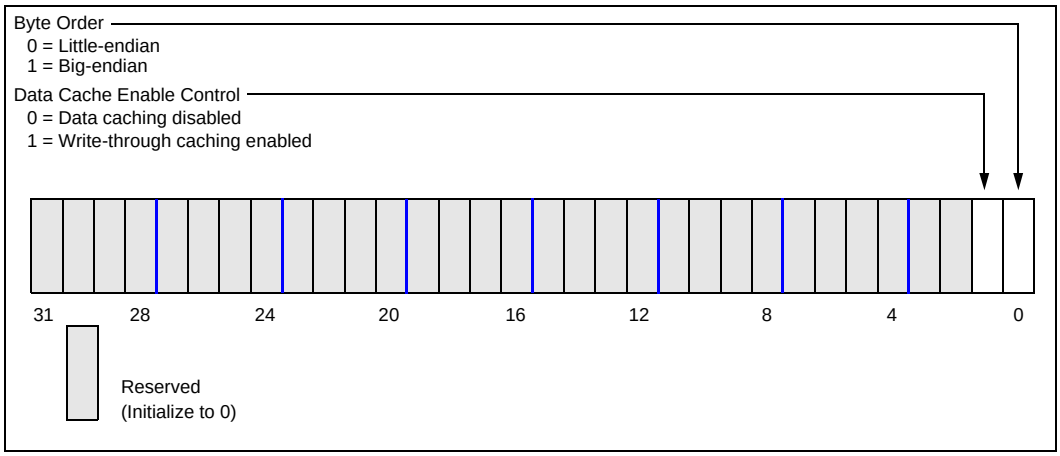

Figure 9. Format of the DLMCON Register on the 80960Jx

The Bus Control Register (BCON) mainly serves to validate the values stored in the PMCON registers (region table). Immediately after a hardware reset, the region table is marked invalid in the BCON register (bit 0 =0). Whenever the region table is marked invalid in the BCON register, the Bus Control Unit (BCU) uses the physical memory parameters found in PMCON14:15 for accesses to all regions (recall that PMCON14:15 is initialized from data found in the IBR). Once the entire region table is loaded by reset microcode, the region table is marked valid in the BCON register (bit 0 = 1), and the BCU uses the appropriate user programmed values. The BCON register format is presented in Figure 10.

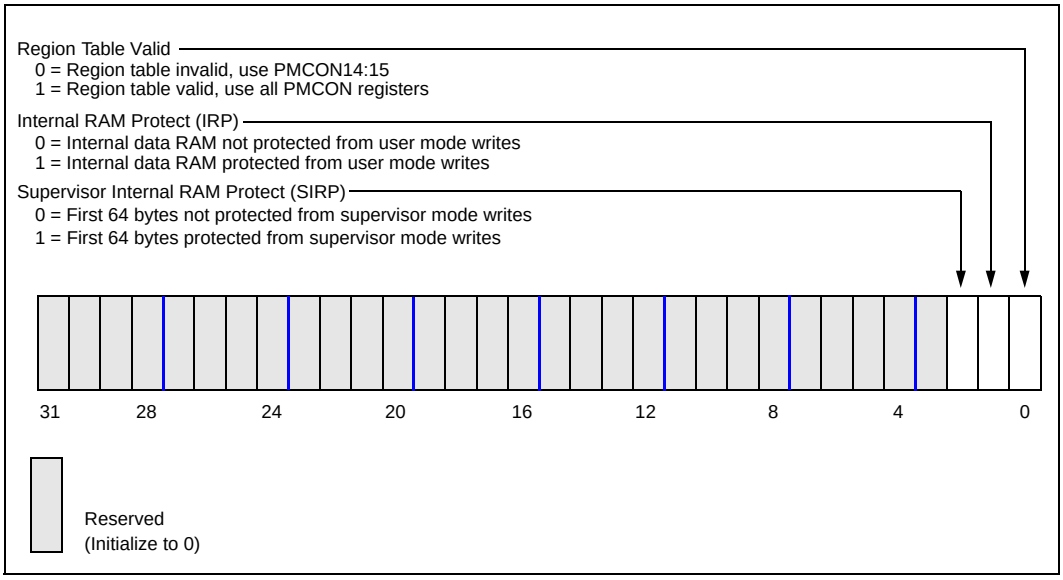

Figure 10. Format of the BCON Register on the 80960Jx

2.3.4.3 External Memory Configuration on the 80960Hx Processor

Control of physical and logical memory characteristics is divided among several registers on the 80960Hx:

- Physical Memory Region Control (PMCON)
- Logical Memory Template Address (LMAR)
- Logical Memory Template Mask (LMMR)
- Default Logical Memory Configuration (DLMCON) registers

There are sixteen PMCON registers, one for each 256 Mbyte physical memory region on the 80960Hx (PMCON0 through PMCON15). There are fifteen pairs of LMAR/LMMR registers (LMAR0/LMMR0 through LMAR14/LMMR14), and one DLMCON register.

PMCON registers define the following physical memory attributes for the 80960Hx.

- Read Address to Data Wait States
- Read Data to Data Wait States
- Write Address to Data Wait States
- Write Data to Data Wait States
- Bus Turnaround Wait States
- Data Bus Parity
- Parity Sense
- Bus Width
- Read Pipelining
- Burst Enable
- $\overline{\text{READY}}$/$\overline{\text{BTERM}}$ Enable

Note that this is a superset of the control found in 80960Cx MCON registers. Also note that the format for the PMCON registers of the 80960Jx and 80960Hx match; 80960Jx PMCON registers select only bus width, but the bit positions are the same.

The PMCON register format is shown in Figure 11.

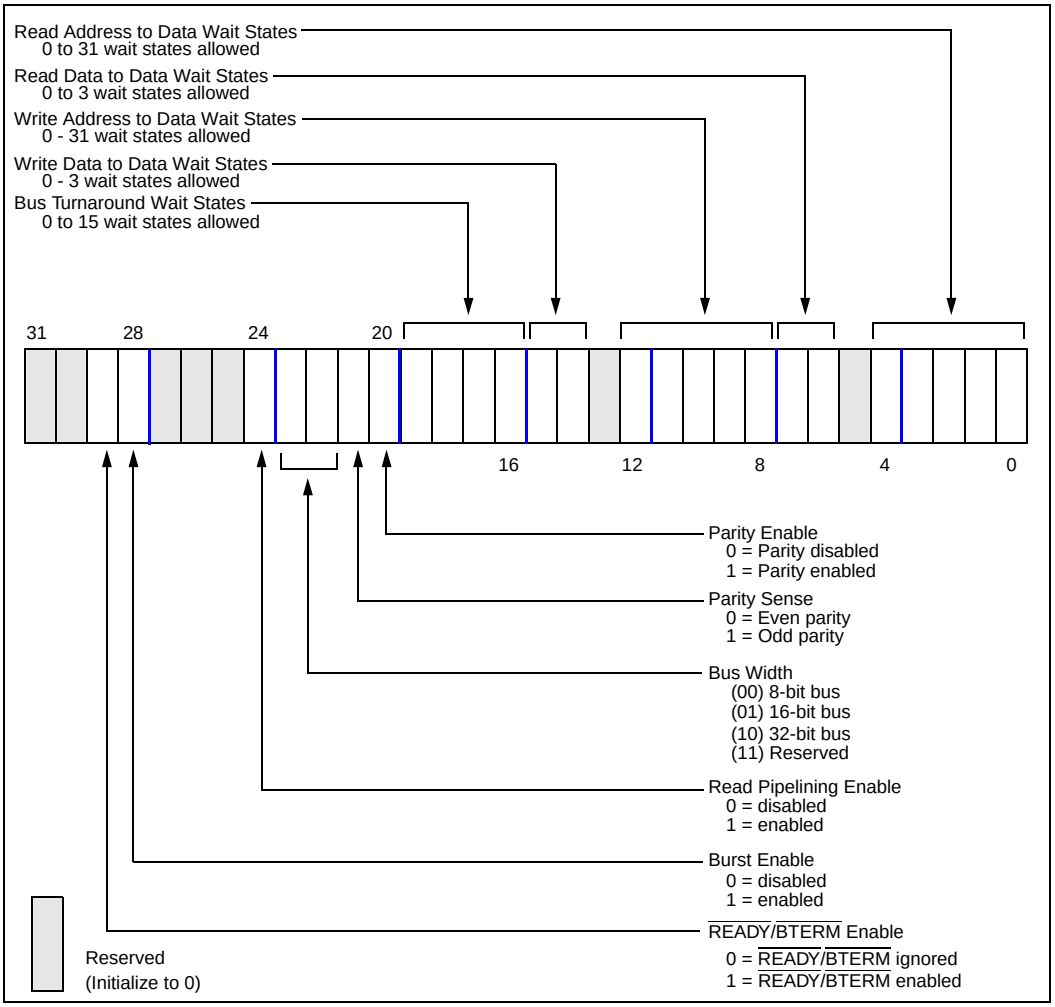

Figure 11. PMCON Register Format for the 80960Hx Processor

Logical memory control registers (LMARx and LMMRx) inform the processor of the logical attributes of external memory. On the 80960Hx, these registers control byte ordering, data caching, and regions to be marked for quick invalidation in the data cache. Lines in the data cache filled from a “quick invalidation” region may be invalidated by executing the Data Cache Control (**dcctl**) instruction, without invalidating the entire cache. Note that byte ordering is controlled by logical region on the 80960Hx; the 80960Jx supports only global byte ordering.

Logical memory regions are created using the LMAR/LMMR register pair. LMAR contains a starting address (4 Kbyte granularity), against which all external accesses are compared. LMMR contains a mask, which indicates which address bits in LMAR are actually compared (compare under mask). In this manner, one register pair may be used to create multiple logical regions, aliased on an arbitrary power-of-two boundary, by making some combination of upper memory bits “don’t care.” LMAR and LMMR register formats are shown in Figure 12.

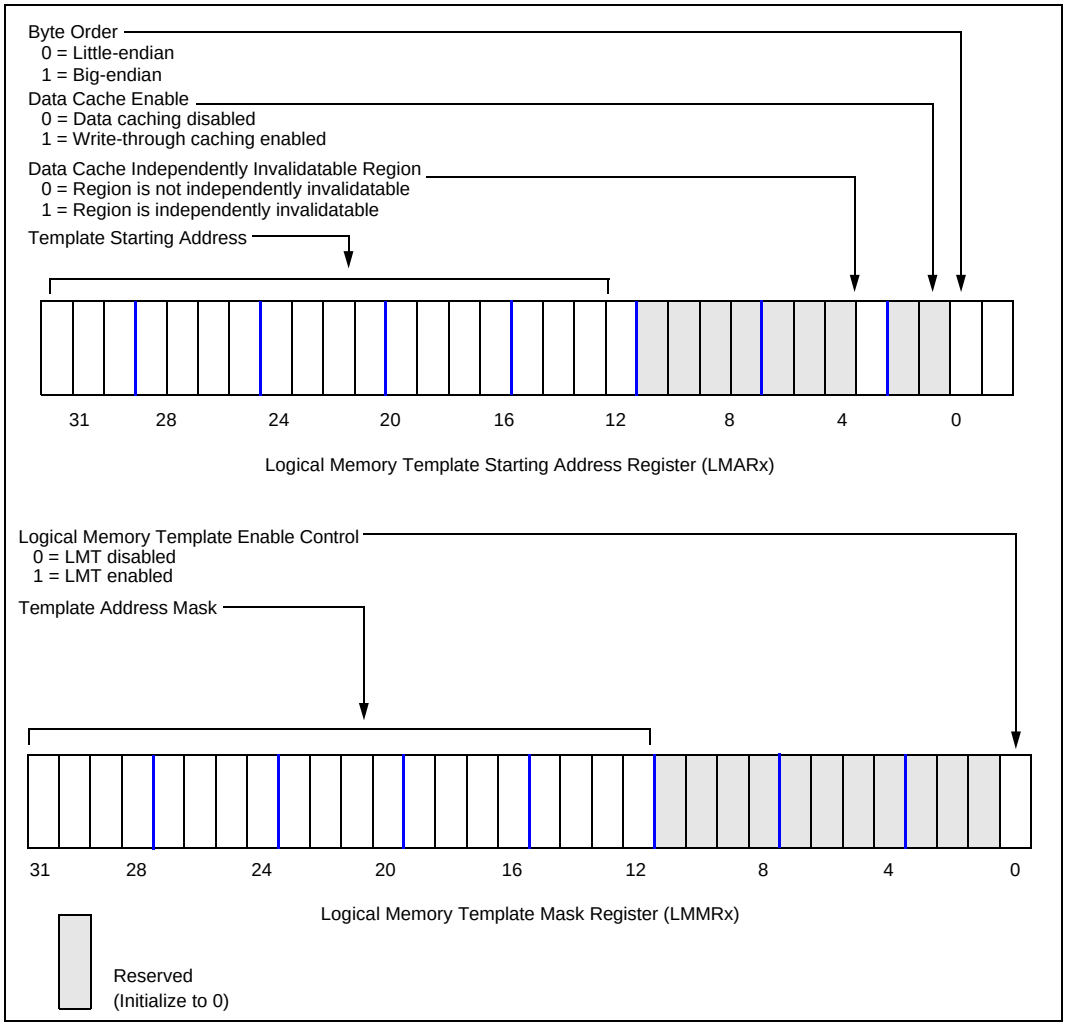

Figure 12. LMARx and LMMRx Registers on the 80960Hx Processor (LMCONx Pair)

The Default Logical Memory Configuration Register (DLMCON) provides default logical memory control for those accesses which do not fall within a region defined by the logical memory control register pairs. Logical attributes controlled by the DLMCON include data caching enable, independent data cache invalidation, and byte ordering. The DLMCON register format is illustrated in Figure 13.

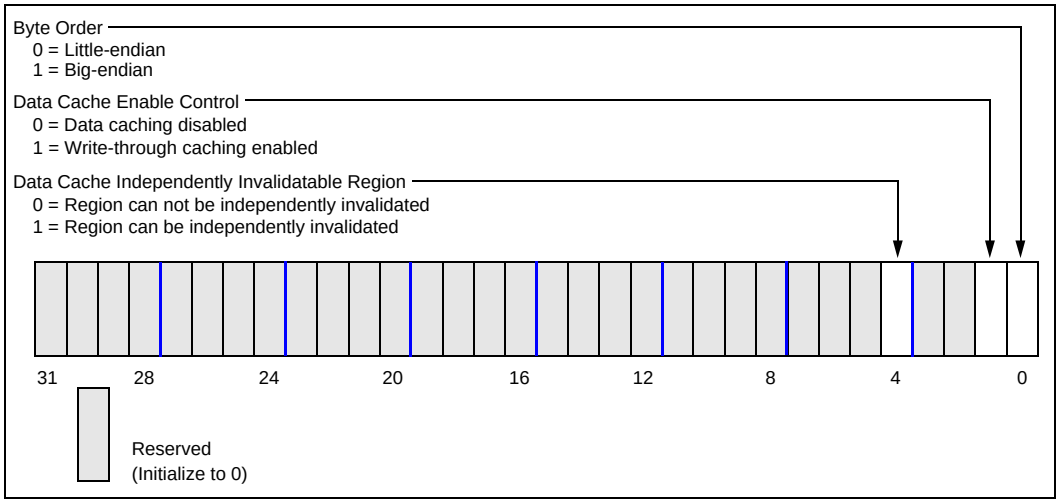

Figure 13. DLMCON Register on the 80960Hx

The Bus Control Register (BCON), represented in Figure 14, mainly serves to validate the values stored in the PMCON registers (region table). Immediately after a hardware reset, the region table is marked invalid in the BCON register (bit 0 = 0). Whenever the region table is marked invalid in the BCON register, the Bus Control Unit (BCU) uses the physical memory parameters found in PMCON15 for accesses to all regions (recall that PMCON15 is initialized from data found in the IBR). Once the entire region table is loaded by reset microcode, the region table is marked valid in the BCON register (bit 0 = 1), and the BCU uses the appropriate user programmed values.

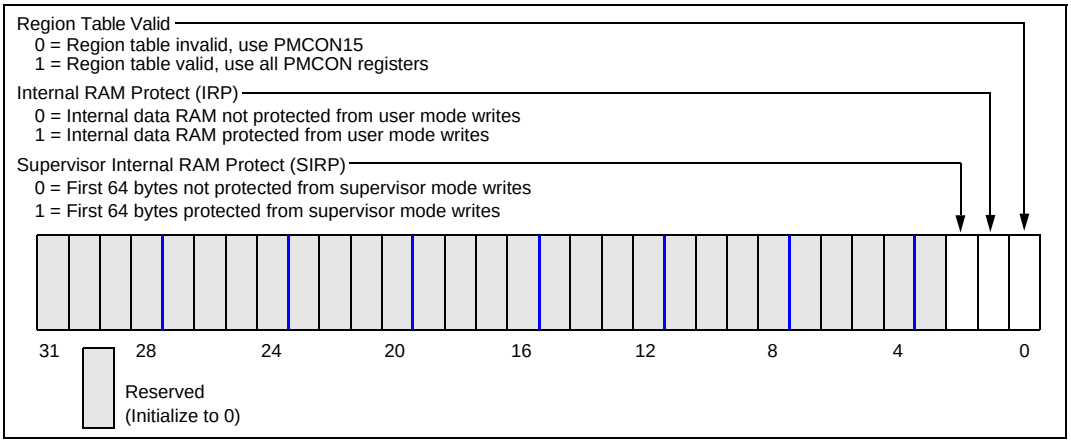

Figure 14. Format of the BCON Register on the 80960Hx

3.0 New and Extended Instructions

The 80960Jx and 80960Hx processors are designed for software upward-compatibility from the 80960Cx processor. The 80960Jx and 80960Hx processors execute any instructions in the 80960Cx instruction set, except the Setup DMA instruction (**sdma**). Executing **sdma** generates a fault. In addition, the processors provide additional instructions not supported by the 80960Cx processor to enhance performance, and provide additional control over on-chip memories and peripherals. These new instructions are summarized in the following sections.

For more detailed information on new instructions, please refer to the appropriate processor users manual.

3.1 New Instructions Supported by the 80960Jx and 80960Hx Processors

Table 6 lists the new 80960Jx and 80960Hx instructions. These instructions are not supported by the 80960Cx processor. These instructions are described in the following sections. Refer to the Jx and Hx users manuals for extensive details.

Table 6. New 80960Jx and 80960Hx Instructions

Instruction Name	Mnemonic	Processor	
		Jx	Hx
Conditional Integer/Ordinal Add	**addi<cc>/addo<cc>**	X	X
Conditional Integer/Ordinal Subtract	**subi<cc>/subo<cc>**	X	X
Byte Swap	**bswap**	X	X
Compare Integer/Ordinal Byte	**cmpib/cmpob**	X	X
Compare Integer/Ordinal Short	**cmpis/cmpos**	X	X
Select Value	**sel<cc>**	X	X
Data Cache Control	**dcctl**	X	X
Instruction Cache Control	**icctl**	X	X
Interrupt Control	**intctl**	X	X
Global Interrupt Disable	**intdis**	X	X
Global Interrupt Enable	**inten**	X	X
Halt	**halt**	X	X
Flush Data Cache Contents by Address	**dcflusha**		X
Give Address to Data Cache as Hint	**dchint**		X
Data Cache Invalidate by Address	**dcinva**		X

NOTES:

1. <cc> indicates conditional instructions. These conditions include: no - unordered, g - greater, e - equal, ge - greater or equal, l - less, le - less or equal, ne - not equal, o - ordered.

3.1.1 Conditional Integer/Ordinal Add and Subtract and Select Value Instructions

Conditional add instructions (**addi**<cc>/**addo**<cc>) provide an efficient method for encoding C language constructs. Consider the following example:

```
if (condition) value += adder; /* may translate to following assembly */
          cmpir4, 0/* condition in r4 */
          addiner5, r6, r5/* value in r5, adder in r6 */
```

This eliminates the need to branch around the addition. A similar example could be cited for the subtract instruction (**subi**<cc>/**subo**<cc>). The select value instruction (**sel**<cc>) provides the ability to conditionally move data, thus an efficient means to encode the following C construct:

```
r = a ? b : c; /* may translate to following assembly */
           cmpir4, 0/* a in r4, tested against 0000.0000H */
           seler5, r6, r7 /* r in r7, b in r5, c in r6 */
```

Again, this eliminates the need to branch around a move instruction.

The operation of these new instructions depends on the condition code mask applied by the instruction, to the current state of the condition code flags. Condition code masks are selected based on the <cc> field, and listed in Table 7.

Table 7. Condition Code Masks

Mask	Instruction	Mask	Instruction
000_2	Unordered	100_2	Less
001_2	Greater	101_2	Not Equal
010_2	Equal	110_2	Less or Equal
011_2	Greater or Equal	111_2	Ordered

NOTES:

1. Mask values correspond to those values applied to the condition code flags, found in the Arithmetic Controls register (AC.cc).

3.1.1.1 Operation of Conditional Add Instruction

The conditional add instructions perform the following operation:

```
if ((((mask != 000₂) && (mask & AC.cc)) != 000₂) || ((mask == 000₂) && (AC.cc == 000₂)))
 src/dst = src2 + src1;
```

3.1.1.2 Operation of Conditional Subtract Instruction

The conditional subtract instructions perform the following operation.

```
if ((((mask != 000₂) && (mask & AC.cc)) != 000₂) || ((mask == 000₂) && (AC.cc == 000₂)))
 src/dst = src2 - src1;
```

3.1.1.3 Operation of Select Value Instruction

The select value performs the following operation:

```
if ((((mask != 000₂) && (mask & AC.cc)) != 000₂) || ((mask == 000₂) && (AC.cc == 000₂)))
 src/dst = src2;
else
 src/dst = src1;
```

3.1.2 Byte Swap Instruction (bswap)

The byte swap instruction is useful for reversing the byte ordering of a value; it swaps a value's endianess from little- to big-endian, or vice versa. For instance, if a register contains FF00.AA55H, then after the **bswap** instruction, it will contain 55AA.00FFH. The operation of this instruction follows:

```
src/dst = ((src1 rotate 8) & 00FF.00FFH) +
((src1 rotate 24) & FF00.FF00H),
```

where rotate means to shift the value to the left by the number of bits specified; bits shifted out of the MSB position are inserted back at the LSB position. The user should be aware that, on some implementations, it is possible to perform the byte-swap operation using an equivalent set of discrete instructions.

3.1.3 Compare Integer/Ordinal Byte and Short Instructions

The compare integer/ordinal byte and short instructions provide for the comparison of 8- and 16-bit quantities, without the need to manually mask upper bits. The operation of these instructions follows:

```
if (src1[N:0] < src2[N:0]) AC.cc = 100₂;
if (src1[N:0] == src2[N:0]) AC.cc = 010₂;
if (src1[N:0] > src2[N:0]) AC.cc = 001₂;
```

where N indicates the number of least-significant bits to compare, and is either seven or fifteen.

3.1.4 Data Cache Control Instruction (dcctl)

The data cache control instruction performs data cache management and control. The *src1* value determines the operation to perform; some operations also use *src2* and *src/dst*. This instruction eliminates the need to provide such functionality in the **sysctl** instruction, thus improving its performance. The operations performed by this instruction are listed in the following sections.

3.1.4.1 Function Zero - Disable the Data Cache (*src1* == 0)

This function globally disables the data cache. Data loads and stores are directed to external memory; no access causes a hit or space to be allocated in the cache. The cache contents remain unchanged.

3.1.4.2 Function One - Enable the Data Cache (*src1* == 1)

This function globally enables the data cache. Data loads may hit in the cache, eliminating the external bus access. Data stores are allocated to the cache, and written to external memory (write through cache).

3.1.4.3 Function Two - Globally Invalidate the Data Cache (*src1* == 2)

This function invalidates the entire data cache.

3.1.4.4 Function Three - Ensure Coherency of the Data Cache with External Memory (*src1* == 3)

This function is meant to ensure that the data cache is guaranteed to be coherent with external memory. In this case, since the data cache is write-through, the cache is invalidated. In this manner, any new data allocated in the cache is coherent with external memory.

3.1.4.5 Function Four - Get Data Cache Status (*src1* == 4)

This function returns the status of the data cache. Status returned in *src/dst* follows the format outlined in Table 8.

Table 8. Format of Data Cache Status

Bits of *src/dst*	Meaning
[0]	Data Cache Enable: 0 = disabled, 1 = enabled
[7:4]	$\log_2$ (bytes per atom)
[11:8]	$\log_2$ (atoms per line)
[15:12]	$\log_2$ (number of sets)
[27:16]	Number of ways minus one

NOTES:

1. Total Cache Size = ([27:16] + 1) << ([7:4] + [11:8] + [15:12])
2. Bits not specified read as zeroes.

3.1.4.6 Function Six - Store Data Cache Sets to Memory (*src1* == 6)

This instruction provides a means to dump the entire data cache, so that its contents may be examined and evaluated. The operation performed by this function follows.

Table 9. Function Six - Store Data Cache Sets to Memory (*src1* == 6)

```
start = src/dst[15:0]; /* starting set number */
end = src/dst[31:16]; /*ending set number */
if (end >= DCACHE_MAX_SETS)
  end = DCACHE_MAX_SETS - 1;
if (start > end)
  GenerateFault(Operation.InvalidOperand);
memadr = src2;
for (set = start; set <= end; set++)
{
  memory[memadr] = SetData[set];
  memadr += 4;
  for (way = 0; way < DCACHE_MAX_WAYS; way++)
  {
    memory[memadr] = tags[set][way];
    memadr += 4;
    memory[memadr] = valid_bits[set][way];
    memadr += 4;
    for (word = 0; word < DCACHE_WORDS_IN_LINE; word++)
    {
      memory[memadr] = DCache_Line[set][way][word];
      memadr += 4;
    }
  }
}
```

3.1.5 Instruction Cache Control Instruction (icctl)

The instruction cache control instruction performs instruction cache management and control. The *sr1* value determines the operation to perform; some operations also use *src2* and *src/dst*. This instruction eliminates the need to provide such functionality in the **sysctl** instruction, thus improving its performance. The operations performed by this instruction are listed in the following subsections.

3.1.5.1 Function Zero - Disable the Instruction Cache (*src*1 == 0)

This function globally disables the instruction cache. Instruction fetches are directed to external memory; no fetch causes a hit or space to be allocated in the cache. The cache contents remain unchanged.

3.1.5.2 Function One - Enable the Instruction Cache (*src1* == 1)

This function globally enables the instruction cache. Instruction fetches may hit in the cache, eliminating the external bus access. Instruction fetches that do not hit are redirected to external memory, and allocated in the cache.

3.1.5.3 Function Two - Globally Invalidate the Instruction Cache (*src1* == 2)

This function invalidates the entire instruction cache.

3.1.5.4 Function Three - Load and Lock Code into the Instruction Cache (*src1* == 3)

This function is used to load and lock instructions into the instruction cache. *Src2* contains the number of contiguous blocks to load; *src/dst* contains the starting address of code to lock. The operation of this function follows:

Table 10. Function Three - Load and Lock Code into the Instruction Cache (*src1* == 3)

```
for (j = 0; j < src2; j++)
{
  way = way_associated_with_block(j);
  start = src/dst + j * block_size;
  end = start + block_size;
  for (i = start; i < end; i += 4)
  {
    set = set_associated_with(i);
    word = word_associated_with(i);
    ICache_Line[set][way][word] = memory[i];
    UpdateTagNValidBits(set, way, word);
    LockICache(set, way, word);
  }
}
```

3.1.5.5 Function Four - Get Instruction Cache Status (*src1* == 4)

This function returns the status of the data cache. Status returned in *src/dst* follows the format outlined in Table 11.

Table 11. Format of Instruction Cache Status

Bits of *src/dst*	Meaning
[0]	Instruction Cache Enable: 0 = disabled, 1 = enabled
[7:4]	$\log_2$ (bytes per atom)
[11:8]	$\log_2$ (atoms per line)
[15:12]	$\log_2$ (number of sets)
[27:16]	Number of ways minus one

NOTES:

1. Total Cache Size = ([27:16] + 1) << ([7:4] + [11:8] + [15:12])
2. Bits not specified read as zeroes.

3.1.5.6 Function Five - Get Instruction Cache Locking Status (*src1* == 5)

This function returns the instruction cache locking status. Status returned in *src/dst* follows the format outlined in Table 12.

Table 12. Format of Instruction Cache Locking Status

Bits of *src/dst*	Meaning
[7:0]	Number of blocks that lock
[23:8]	Block size in words
[31:24]	Number of locked blocks

3.1.5.7 Function Six - Store Instruction Cache Sets to Memory (*src1* == 6)

This instruction provides a means to dump the entire instruction cache, so that its contents may be examined and evaluated. The operation performed by this function follows.

Table 13. Function Six - Store Instruction Cache Sets to Memory (*src1* == 6)

```
start = src/dst[15:0]; /* starting set number */
end = src/dst[31:16]; /*ending set number */
if (end >= ICACHE_MAX_SETS)
  end = ICACHE_MAX_SETS - 1;
if (start > end)
  GenerateFault(Operation.InvalidOperand);
memadr = src2;
for (set = start; set <= end; set++)
{
  memory[memadr] = SetData[set];
  memadr += 4;
  for (way = 0; way < ICACHE_MAX_WAYS; way++)
  {
    memory[memadr] = tags[set][way];
    memadr += 4;
    memory[memadr] = valid_bits[set][way];
    memadr += 4;
    for (word = 0; word < ICACHE_WORDS_IN_LINE; word++)
    {
      memory[memadr] = ICache_Line[set][way][word];
      memadr += 4;
    }
  }
}
```

3.1.6 Interrupt Control Instruction

This instruction provides a means of globally enabling or disabling interrupts; the previous state of the interrupt controller is returned. The interrupt control instruction may also be used to interrogate the state of the interrupt controller, whether enabled or disabled. The processor guarantees that the new interrupt controller state is in effect before the instruction completes.

3.1.6.1 Function Zero - Globally Disable Interrupts (*src1* = 0)

Executing this function globally disables interrupts by setting the global interrupt enable bit of the Interrupt Control register (ICON.gie = 1). The previous state of ICON.gie is returned in bit zero of the *src/dst*.

3.1.6.2 Function One - Globally Enable Interrupts (*src1* = 1)

Executing this function globally enables interrupts by clearing the global interrupt enable bit of the Interrupt Control register (ICON.gie = 0). The previous state of ICON.gie is returned in bit zero of the *src/dst*.

3.1.6.3 Function Two - Return Interrupt Controller Status (*src1* = 2)

By executing this function, the state of the global interrupt enable bit of the Interrupt Control register (ICON.gie) is returned in bit zero of *src/dst*.

3.1.7 Global Interrupt Disable

This instruction provides a high-performance method for globally disabling interrupts. Interrupts are disabled by setting the global interrupt enable bit in the Interrupt Control register (ICON.gie = 1). In timing-critical routines, this instruction is preferable to the Interrupt Control instruction (**intctl**).

3.1.8 Global Interrupt Enable

This instruction provides a high-performance method for globally enabling interrupts. Interrupts are enabled by clearing the global interrupt enable bit in the Interrupt Control register (ICON.gie = 0). In timing-critical routines, this instruction is preferable to the Interrupt Control instruction (**intctl**).

3.1.9 Halt Instruction

The **halt** instruction provides a means for the processor to prepare itself to receive an interrupt, and to service that interrupt quickly. Once the **halt** instruction executes processing ceases until an interrupt is received of sufficient priority to cause an exit from the halted state. Such an interrupt is serviced, and then normal execution continues at the instruction after **halt**. The **halt** instruction may enable or disable interrupts based on the value of *src1*.

3.1.9.1 Function Zero - Globally Disable Interrupts (*src1* = 0)

When this instruction executes with *src1* equal to zero, the global interrupt enable bit of the Interrupt Control register is set (ICON.gie = 1). This masks all interrupts except a non-maskable interrupt (NMI); only an NMI event can bring the processor out of the halted state in this case.

3.1.9.2 Function One - Globally Enable Interrupts (*src1* = 1)

When this instruction executes with *src*1 equal to one, the global interrupt enable bit of the Interrupt Control register is cleared (ICON.gie = 0). This globally enables all interrupts; any interrupt of sufficient priority causes the processor to exit halted state in favor of servicing the interrupt.

3.1.9.3 Function Two - Halt Without Modifying the Current Interrupt State (*src1* = 2)

When this instruction executes with *src1* equal to two, the interrupt controller state remains unmodified. The global interrupt enable state of the processor is not affected.

3.1.10 Flush Data Cache Contents by Address Instruction (dcflusha)

If the effective address referenced by this instruction hits in the data cache, then steps must be taken to guarantee that the value cached for any part or all of the quad-word that the address specifies must be the same as the values for that quad-word in external memory. Since the 80960Hx incorporates a write-through cache, this instruction performs the same operation as the **dcinva** instruction; the quad-word of data in which the address falls is invalidated.

This instruction is available only on the 80960Hx processor.

3.1.11 Give Address to Data Cache as Hint Instruction (dchint)

This instruction is architecturally specified as either a hint to the data cache, or as a no-operation.

If implemented as a data cache hint, it generates an effective address that would be sent to the data cache, indicating that data at the specified address is likely to be needed soon. If this hint to the data cache results in a fault generation, the hint is ignored and no fault is generated.

This instruction is available only on the 80960Hx processor, and is implemented as a no-operation.

3.1.12 Data Cache Invalidate by Address Instruction (dcinva)

The effective address referenced in this instruction is sent to the data cache. The cache must guarantee that after execution of this instruction, the quad-word of data in which this effective address falls is not cached.

This instruction is available only on the 80960Hx processor.

3.2 Extended Instructions

The 80960Jx and 80960Hx processors extend the functionality of the **sysctl** instruction, as originally supported by the 80960Cx processor. This additional functionality supports access to memory-mapped control registers, and the breakpoint sharing mechanism.

3.2.1 System Control Instruction (sysctl)

The system control (**sysctl**) instruction present on the 80960Cx processor has expanded functionality on the 80960Jx and 80960Hx processors. The function that **sysctl** performs is determined from the message type of the instruction. The message type is obtained by logically ANDing *src1* with FF00H, and shifting the results to the right by eight bits ((*src1* & FF00H) >> 8). The various functions of the **sysctl** instruction are detailed in the following sections.

3.2.1.1 Function Zero - Post Software Interrupt (Message Type = 00H)

This function behaves identically to the 80960Cx processor. Typically, **sysctl** is used to request an interrupt in a program. The request interrupt message type (00H) is selected and the interrupt procedure pointer number is specified in the least significant byte of the instruction operand.

3.2.1.2 Function One - Invalidate the Instruction Cache (Message Type = 01H)

This function behaves identically to the 80960Cx processor.

3.2.1.3 Function Two - Configure the Instruction Cache (Message Type = 02H)

The 80960Cx processor supports four modes of operation: modes 0, 1, 4, and 6. The mode is determined by logically ANDing *src1* with FFH (*src1* & FFH). Mode 0 and mode 1 enable and disable the instruction cache, respectively. Mode 4 may be used to load and lock the entire instruction cache (1 Kbyte), while mode 6 is used to load and lock only half the instruction cache (512 bytes).

The 80960Jx and 80960Hx processors also support four modes of operation with slightly different behavior; they also are modes 0, 1, 4, and 6. Again modes 0 and 1 may be used to enable and disable the instruction cache, respectively. However, modes 4 and 6 differ slightly from the 80960Cx processor operation. These two modes are used to load and lock the instruction cache; the number of blocks (ways) to load and lock are indicated by *src2*. The algorithm used to load and lock the cache is the same as that outlined for the instruction cache control instruction (**icctl**).

3.2.1.4 Function Three - Software Reinitialization (Message Type 03H)

This function behaves identically to the 80960Cx processor. The 80960Jx and 80960Hx processors guarantee that the instruction and data caches are both invalidated and disabled before execution resumes.

3.2.1.5 Function Four - Load One Group of Control Registers (Message Type 04H)

This function is implemented on the 80960Cx processor. The 80960Jx and 80960Hx processors do not support this function.

3.2.1.6 Function Five - Modify One Memory-mapped Control Register (Message Type 05H)

This function provides a means to modify a specified memory-mapped control register. The upper sixteen bits of *src1* (*src1*[31:16]) contain the two low-order bytes of the address of the memory-mapped register to access. The new value to write is contained in *src2*, and *src3* contains a mask. After the operation completes, the previous value of the memory-mapped register is returned in *src/dst*. The operation performed by this function follows.

```
addr = (FF00H << 16) | (src1 >> 16);
temp = memory[addr];
memory[addr] = (src2 & src/dst)
| (temp & ~src/dst);
src/dst = temp;
```

This function is **not** supported on the 80960Cx processor.

3.2.1.7 Function Six - Breakpoint Resource Request (Message Type 06H)

To properly use the hardware breakpoint resources of the 80960Jx and 80960Hx, application code must first request access to these resources. To do so, the application must execute this function of the **sysctl** instruction, and use this function. After completion, the value of *src/dst* indicates the breakpoint resources available to the application. This is shown in Table 14. For more information on breakpoint resource sharing, refer to section 5.0, Breakpoint Resource Sharing Mechanism.

Table 14. *src/dst* Field Definitions for Breakpoint Resource Request

Bits of *src/dst*	Description
[3:0]	Number of available instruction breakpoint registers
[7:4]	Number of available data address breakpoint registers
[31:8]	Read as zeroes

This function is **not** supported on the 80960Cx processor.

4.0 Register Cache/Stack Frames

The 80960Cx, 80960Jx, and 80960Hx processors each implement a register cache to store local register sets during explicit and implicit calls. By definition, this memory is not a cache, but a FIFO; the last register set stored in the memory is the first to be retrieved. The register cache is configured from the Register Cache Configuration Word (RCCW), which is read from the PRCB during initialization (see section 2.1.2 Processor Control Block (PRCB)). This register's format and control functions of vary from processor to processor, as discussed in the following sections.

4.1 Register Cache Configuration on the 80960Cx Processor

The 80960Cx processor provides for up to fifteen local register sets to be stored on-chip. If more than five registers sets are specified, additional storage space is taken from internal data RAM, starting at the highest available location and working down from there. For instance, if storage is requested for six register sets, the register cache stores the first five, and data RAM locations 0000.03C0H through 0000.03FFH are used to store the sixth set (recall the 80960Cx provides 1 Kbyte of on-chip data RAM located from addresses 0000.0000H through 0000.03FFH). The user code initializes the least-significant four bits of the RCCW with the total number of register sets to be cached on-chip. If greater than five sets are requested, the required amount of on-chip data RAM is used as described above. The format of the RCCW is shown in Figure 15.

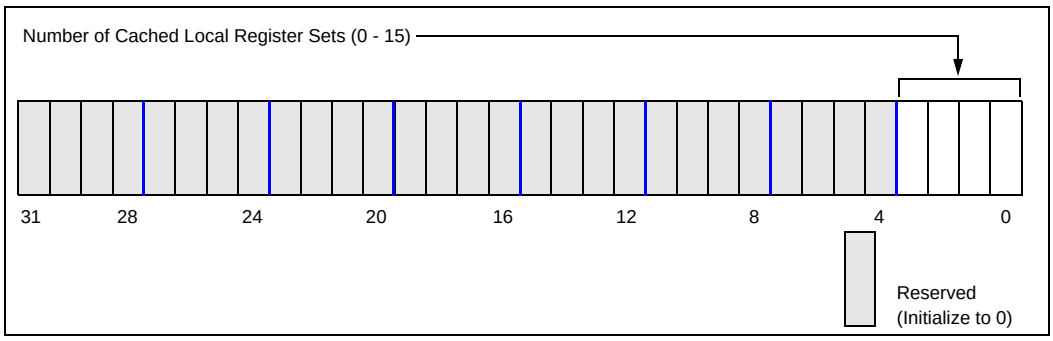

Figure 15. Register Cache Configuration Word (RCCW) for the 80960Cx Processor

4.2 Register Cache Configuration on the 80960Jx Processor

The 80960Jx processor provides for eight local register sets to be stored in the register cache. Unlike the 80960Cx and 80960Hx processors, additional register frames cannot be stored in the register cache by consuming additional on-chip data RAM.

By initializing the RCCW, application code may reserve register frames within the register cache for use only by high-priority interrupts (priority 28 or above). This is accomplished by programming bits 10 through 8 with the appropriate value. Allowed values of the programmed limit range from 000_2 to 111_2. Setting the value to 111_2 reserves all register frames for high-priority interrupts. Setting the value to 000_2 reserves no register frames for high-priority interrupts.

The RCCW is initialized from the PRCB during initialization. It is not accessible through the memory-mapped control register interface. Register format is shown in Figure 16.

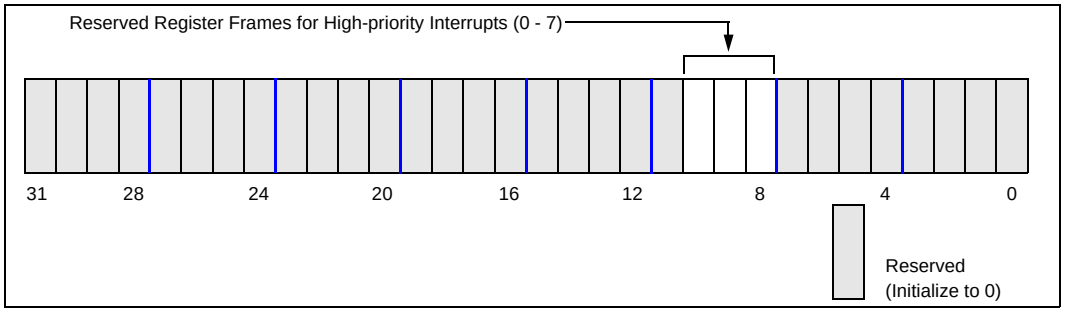

Figure 16. Register Cache Configuration Word (RCCW) for the 80960Jx Processor

4.3 Register Cache Configuration on the 80960Hx Processor

The 80960Hx processor provides for up to fifteen local register sets to be stored on-chip. If more than five register sets are specified, additional storage space is taken from internal data RAM, starting at the highest available location and working down from there. For instance, if storage is requested for six register sets, the register cache stores the first five, and data RAM locations 0000.07C0H through 0000.07FFH are used to store the sixth set (recall the 80960Hx provides 2 Kbytes of on-chip data RAM located from addresses 0000.0000H through 0000.07FFH).

User code initializes the least-significant four bits of the RCCW with the total number of register sets to be cached on-chip. If greater than five sets are requested, the required amount of on-chip data RAM is used as described above. By initializing

the RCCW appropriately, application code may reserve register frames within the register cache for use only for high-priority interrupts (priority 28 or above). This is accomplished by programming bits 12 through 8 with the appropriate value. Allowed values of the programmed limit range from 0000_2 to 1111_2. Setting the value to 1111_2 reserves all register frames for high-priority interrupts. Setting the value to 0000_2 reserves no register frames for high-priority interrupts. The number of frames reserved for high-priority interrupts must be equal to or less than the total number of frames available.

The RCCW is initialized from the PRCB during initialization. It is not accessible through the memory-mapped control register interface. The register format is shown in Figure 17.

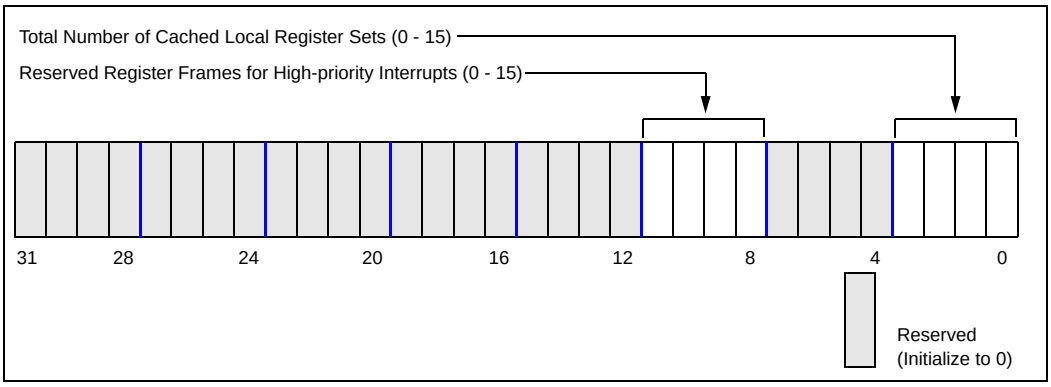

Figure 17. Register Cache Configuration Word (RCCW) for the 80960Hx Processor

5.0 Breakpoint Resource Sharing Mechanism

The 80960 family of processors provide significant on-chip debugging capabilities, including breakpoint and breakpoint control registers. While breakpoint resources included on the 80960Cx, 80960Jx, and 80960Hx processors are similar in function, the access model is different. Accesses to breakpoint resources on the 80960Jx and 80960Hx processors are controlled such that contention between application code (such as embedded debuggers) and hardware development tools does not occur. This section discusses these differences.

5.1 Breakpoint Resources on the 80960Cx Processor

The 80960Cx processor provides two instruction address (IPB0, IPB1) and two data address (DAB0, DAB1) breakpoint registers. In addition, the Breakpoint Control (BPCON) register serves to control the data address breakpoint registers. The 32-bit IPBx registers contain the 30-bit address on which to break normal execution. Only 30 bits of address are compared, because instruction addresses must be word aligned. The low-order bit of each register provides enable/disable control for each register. Execution "breaks" after the instruction executes. The DABx registers contain the 32-bit address at which an appropriate data access causes the processor to "break" normal execution. Data address breakpoints are controlled by the BPCON register, and may be configured to break on:

1. stores
2. loads or stores
3. load, store, or instruction fetch
4. any access

Application code programs these registers by using the "load control register" function of the **sysctl** instruction. The only restriction here is that the processor must be in supervisor mode to execute the **sysctl** instruction. These registers are also initialized at reset and software reinitialization, from values contained in the control table. There is no controlled access or sharing mechanism provided with the 80960Cx processor.

5.2 Breakpoint Resources on the 80960Jx Processor

The 80960Jx processor provides two instruction address (IPB0, IPB1) and two data address (DAB0, DAB1) breakpoint registers. In addition, the Breakpoint Control (BPCON) register serves to control the data address breakpoint registers. The 32-bit IPBx registers contain the 30-bit address on which to break normal execution. Only 30 bits of address are compared, because instruction addresses must be word aligned. The low-order bit of each register provides enable/disable control for each register. Execution "breaks" after the instruction executes. The DABx registers contain the 32-bit address at which an appropriate data access causes the processor to break normal execution. Data address breakpoints are controlled by the BPCON register, and may be configured to break under the following conditions:

1. stores
2. loads or stores
3. loads

These registers are accessed through the **sysctl** instruction, or as memory-mapped control registers.

To avoid contention between hardware development tools and application code, the 80960Jx implements a breakpoint sharing mechanism. Application code must always first request and acquire hardware breakpoint resources before any attempt is made to access them. Hardware development tools exercise supervisor control over breakpoint resource allocation. If the tool retains control of breakpoint resources, none are available for application code. If a development tool is not being used, access privilege to access breakpoint resources is granted to the application. The development tool may relinquish control of breakpoint resources to the application at its discretion.

If the application attempts to access the breakpoint or breakpoint control (BPCON) registers without first acquiring access rights to them, a fault is generated. In this case, the breakpoint resource is not modified, whether accessed through a **sysctl** instruction or as a memory-mapped register. These breakpoint registers are not initialized from the control table at reset or software reinitialization. Instead, they are initialized during the execution of the **sysctl** instruction when access is requested, only when the application successfully gains access to these registers.

For more information on breakpoint resource sharing, refer to the *i960® Jx Microprocessor User's Manual*.

5.3 Breakpoint Resources on the 80960Hx Processor

The 80960Hx processor provides six instruction address (IPB0 - IPB5) and six data address (DAB0 - DAB5) breakpoint registers. In addition, the Breakpoint Control (BPCON) and Expanded Breakpoint Control (XBPCON) registers serve to control the data address breakpoint registers. The 32-bit IPBx registers contain the 30-bit address on which to break normal execution. Only 30 bits of address are compared, because instruction addresses must be word aligned. The low-order bit of each register provides enable/disable control for each register. Execution breaks after the instruction executes. The DABx registers contain the 32-bit address at which an appropriate data access causes the processor to break normal execution. Data address breakpoints are controlled by the BPCON register, and may be configured to break under the following conditions:

1. stores
2. loads or stores
3. loads

These registers are accessed through the **sysctl** instruction, or as memory-mapped control registers.

To avoid contention between hardware development tools and application code, the 80960Hx also implements a breakpoint sharing mechanism. Application code must always first request and acquire hardware breakpoint resources before any attempt is made to access them. Hardware development tools exercise supervisor control over breakpoint resource allocation. If the tool retains control of breakpoint resources, none are available for application code. If the tool is not being used, access privilege to access breakpoint resources are granted to the application. It is possible on the 80960Hx for the application to obtain legal access to IPB0, IPB1, DAB0, and DAB1, and for the remaining breakpoint resources to remain under control of the development tool. Like the 80960Jx, the tool may relinquish control of breakpoint resources to the application at its discretion.

If the application attempts to access the breakpoint or breakpoint control (BPCON) registers without first acquiring access rights to them, a fault is generated. In this case, the breakpoint resource is not modified, whether

accessed through a **sysctl** instruction or as a memory-mapped register. These breakpoint registers are not initialized from the control table at reset or software reinitialization. Instead, they are initialized during the execution of the **sysctl** instruction when access is requested, only when the application successfully gains access to these registers.

For more information on breakpoint resource sharing, refer to the *i960®Hx Microprocessor User's Manual*.

6.0 Integrated Peripherals

The integrated peripheral sets available on the 80960Cx, 80960Jx, and 80960Hx processors are summarized in Table 15.

Table 15. Integrated Peripherals on the 80960Cx, 80960Jx, and 80960Hx Processors

Peripheral	Processor		
	80960Cx	80960Jx	80960Hx
Direct Memory Access Controller	Yes	No	No
Interrupt Controller	Yes	Yes	Yes
Guarded Memory Unit	No	No	Yes
Timers	None	Two	Two

6.1 Direct Memory Access Control Unit on the 80960Cx Processor

The Direct Memory Access controller on the 80960Cx processor provides integrated DMA capability to the application. The DMA controller can manage four independent DMA channels concurrently. Each channel supports transfers between any combination of external memory and internal data RAM. There are two modes employed to accomplish DMA transfers:

- demand mode (synchronous)
- block mode (asynchronous)

Demand mode transfers typically move data between external devices and memory. Block mode transfers typically move blocks of data within memory.

The 80960Cx processor provides a 13-pin interface to manage DMA transfers. These pins include:

- DMA request (DREQx)
- DMA acknowledge (DACKx)
- End of Process/Terminal Count (EOPx/TCx), and
- DMA Access (DMA)

The 80960Cx DMA controller provides two means to transfer data: standard and fly-by transfers. During standard transfers, multiple bus cycles are generated until the transfer is complete. During fly-by, the processor uses idle time on the bus to "slip in" a DMA bus cycle. Transfers may occur between any combination of 8-, 16-, and 32-bit memories using byte, short, word, or quad-word transactions.

DMA functionality is controlled by programming the DMA Command (DMAC) register, and the DMA Control Word. A special instruction, the Set Up DMA (**sdma**) instruction is also used to configure DMA functionality.

Programming the DMA controller is not discussed in detail in this document. Using the DMA capabilities of the 80960Cx processor precludes an application from being 80960Hx-ready. The user must not use DMA functions of the 80960Cx processor if the design will be upgraded to the 80960Hx processor. For more information on the DMA controller, refer to the *i960® Cx Microprocessor User's Manual*.

6.2 Guarded Memory Unit on the 80960Hx Processor

The Guarded Memory Unit (GMU) provides memory protection capabilities to the 80960Hx processor. This is useful during software development to detect and protect against illegal memory accesses. The GMU does not degrade execution, or in any way affect correct program execution. This peripheral is available only on the 80960Hx processor.

Using the GMU, application code may set up ranges of addresses with programmable access privileges. These privileges include: user read, user write, user execute, supervisor read, supervisor write, and supervisor execute. The GMU provides two range types: *protection* and *detection* ranges. Illegal accesses which fall within a protected range are prevented from altering memory. The processor guarantees that the access does not result in an internal cache or data RAM access, or generate an external bus cycle. The `PROTECTION.BAD_ACCESS` fault is also generated. An unauthorized access which falls within a detection range is allowed to complete, but the same protection fault is generated. An example of how the GMU might be configured to provide memory protection is given in Table 16.

Table 16. Sample Application with Restricted Memory Partitions

Application Memory Partition	Supervisor		User		Execute	
	Read	Write	Read	Write	Read	Write
Application/User RAM			X	X		
User Mode Stack	X	X	X	X		
Interrupt, Supervisor, Fault Tables	X		X			
Interrupt Stack	X	X				
Supervisor Stack	X	X				
Kernel RAM	X	X				
Kernel Code and Application Code					X	X

NOTES:

1. An ‘X’ in a location indicates the access is allowed.

The GMU is configured using several control registers:

- GMU Control register (GCR)
- GMU Memory Protect Address register pairs (MPARx, MPMRx), and
- GMU Memory Violation Detection register pairs (MDUBx, MULBx)

Two memory protect address register pairs are provided, allowing for a minimum of two protection ranges to be selected. More than two protection ranges are possible because the protection range address pairs are implemented as a base address and mask. An arbitrary number of addresses may be aliased by setting the appropriate mask bits. A total of six memory violation detection ranges may be selected. These register pairs act as true range (upper and lower bounds) registers.

Programming the GMU is not discussed in detail in this document. For more information on the GMU, refer to the *i960® Hx Microprocessor User’s Manual*.

6.3 Interrupt Control Unit

The 80960Cx, 80960Jx, and 80960Hx processors implement an integrated interrupt controller. The interrupt controllers present on the 80960Jx and 80960Hx are identical. The 80960Jx and 80960Hx implementations draft directly from the 80960Cx. The following sections detail differences between the 80960Cx and the 80960Jx/ 80960Hx interrupt controllers.

6.3.1 80960Cx Processor Interrupt Control Unit

The 80960Cx processor’s interrupt controller is configured by programming the Interrupt Control (ICON), Interrupt Mapping (IMAP0 - IMAP2), Interrupt Mask (IMSK), and Interrupt Pending (IPND) registers. The formats of these registers are given in the following paragraphs.

6.3.1.1 Interrupt Control Register (ICON)

The interrupt control register of the 80960Cx processor (see Figure 18) controls basic functionality of the interrupt controller such as interrupt mode, signal detection, global enable/disable, mask operation, interrupt vector caching, and sampling mode.

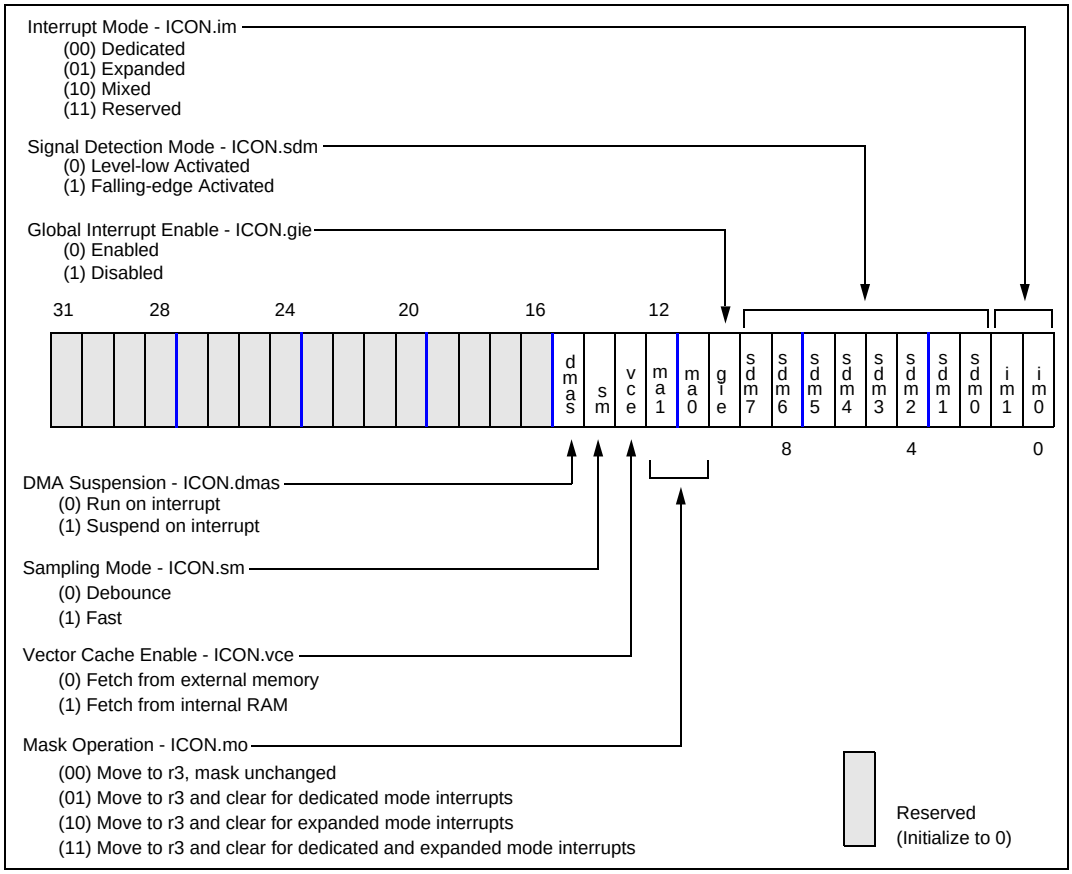

Figure 18. Interrupt Control Register on the 80960Cx Processor

Note the control bits designated in Figure 18 which are related to DMA functionality. The 80960Jx and 80960Hx processors do not support DMA capability, and therefore do not contain these control bits. Programming the ICON register is supported through the **sysctl** instruction on the 80960Cx processor.

6.3.1.2 Interrupt Mapping Registers (IMAP0-IMAP2)

Interrupt mapping registers are used to program the vector number associated with an interrupt source when the source is connected to a dedicated-mode input.

- Registers IMAP0 and IMAP1 contain mapping information for the external interrupt pins (four bits per pin
- Register IMAP2 contains mapping information for the DMA interrupt inputs (four bits per input)

The format of these registers is shown in Figure 19.

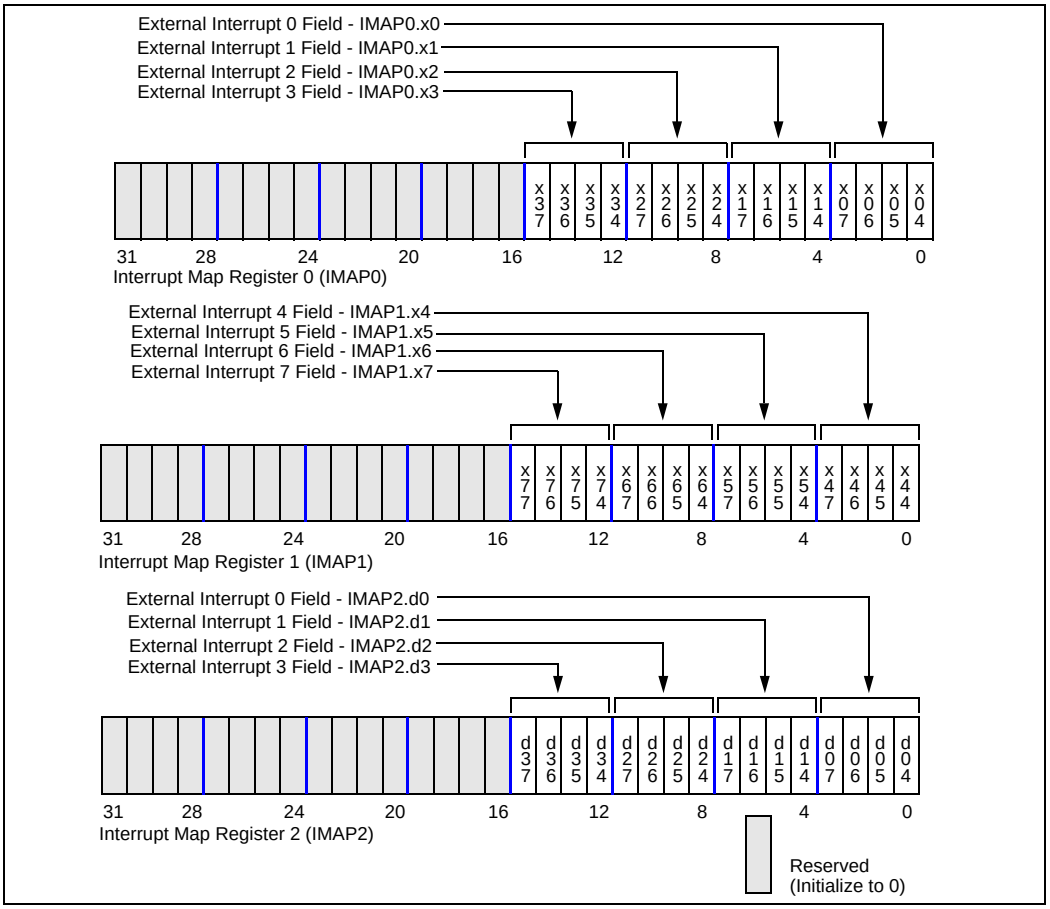

Figure 19. Interrupt Mapping Registers on the 80960Cx Processor

Note the DMA-related control bits. The 80960Jx and 80960Hx processors do not support DMA functionality; therefore these control bits do not exist. The IMAPx registers are programmed by using the **sysctl** instruction.

6.3.1.3 Interrupt Mask and Interrupt Pending Registers (IMSK, IPND)

The interrupt mask register provides a mechanism to uniquely mask dedicated mode interrupts and DMA interrupt sources, or globally mask expanded mode interrupts. In this manner, application software may prevent servicing of these interrupts. The interrupt pending register serves as a means to post dedicated mode interrupts, or interrupts from a DMA source. Application code which sets a bit in the interrupt pending register precipitates the same action that would occur if an external interrupt source caused the bit to be set.

These registers are accessed as special function registers: interrupt pending as sf0, interrupt mask as sf1. On reset, the interrupt mask is cleared (= 0) and interrupt pending undefined. Application code is responsible for explicitly initializing these registers. Figure 20 shows the format of these registers.

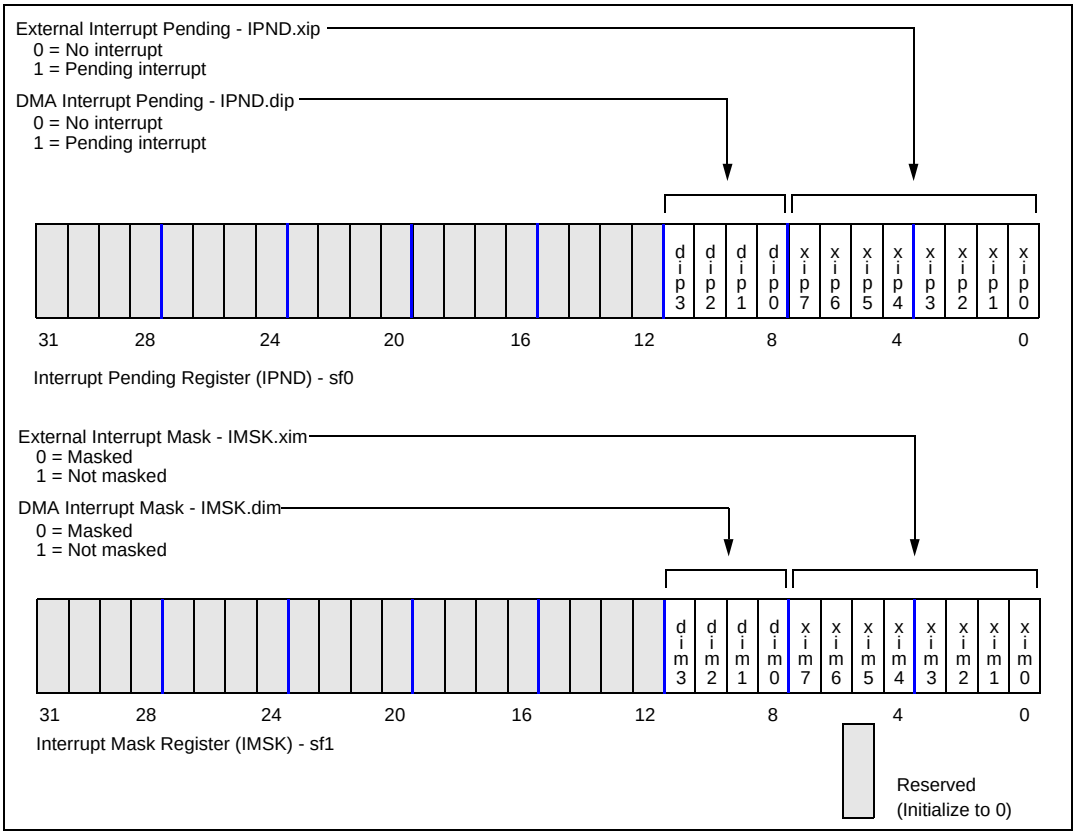

Figure 20. Interrupt Pending (sf0) and Interrupt Mask (sf1) on the 80960Cx Processor

6.3.2 80960Jx/80960Hx Processor Interrupt Control Unit

The 80960Jx and 80960Hx interrupt controllers are configured by programming the Interrupt Control (ICON), Interrupt Mapping (IMAP0 - IMAP2), Interrupt Mask (IMSK), and Interrupt Pending (IPND) registers. The formats of these registers are given in the following paragraphs.

6.3.2.1 Interrupt Control Register (ICON)

The interrupt control register of the 80960Jx and 80960Hx processors control basic functionality of the interrupt controller such as interrupt mode, signal detection, global enable/disable, mask operation, interrupt vector caching, and sampling mode. The format of this register is shown in Figure 21.

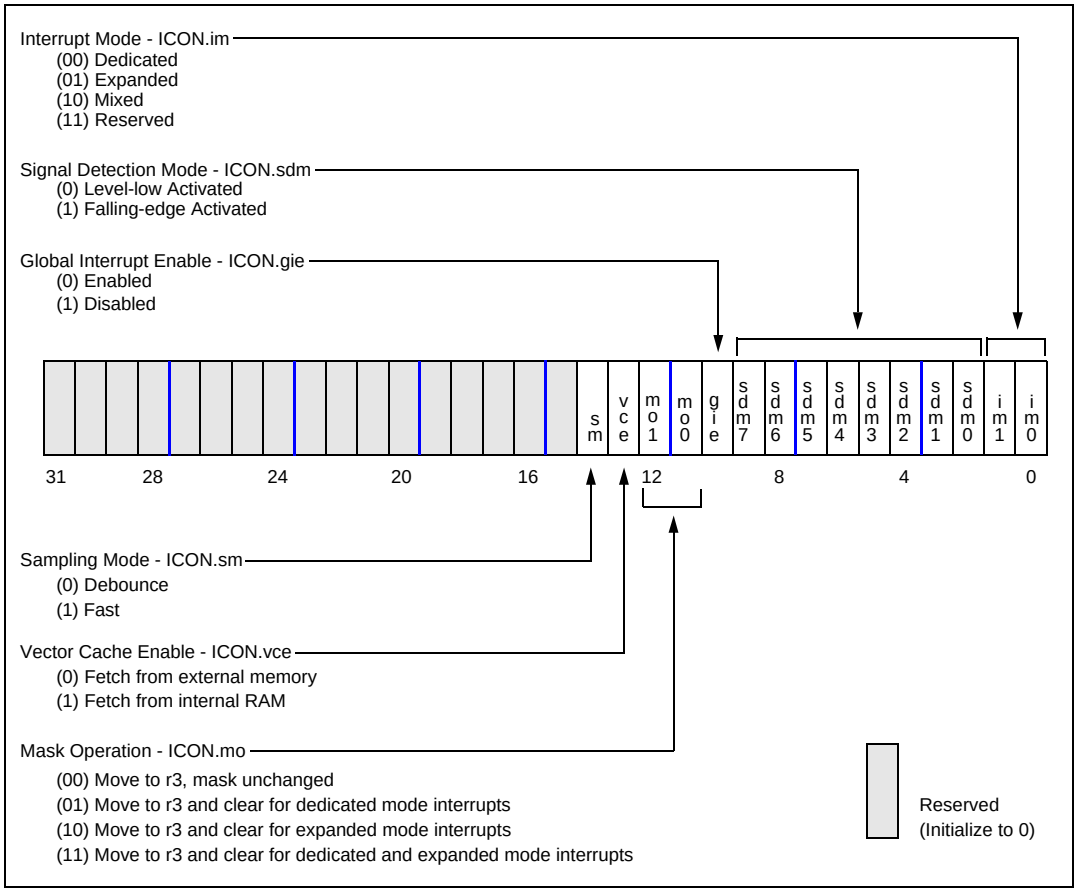

Figure 21. Interrupt Control Register (ICON) on the 80960Jx and 80960Hx Processors

Programming the ICON register is supported through the **sysctl** instruction. The ICON register is also mapped to the memory-mapped register space, address FF00.8510H, and may be accessed as such in supervisor mode. This register is also affected by the interrupt control (**intctl**), interrupt disable (**intdis**), and interrupt enable (**inten**) instructions. Refer to section 3.0 New and Extended Instructions, for more information.

6.3.2.2 Interrupt Mapping Registers (IMAP0-IMAP2)

Interrupt mapping registers are used to program the vector number associated with an interrupt source when the source is connected to a dedicated-mode input, or timer interrupt.

- Registers IMAP0 and IMAP1 contain mapping information for the external interrupt pins (four bits per pin)
- register IMAP2 contains mapping information for the timer interrupt sources (four bits per input)

The format of these registers is shown in Figure 22.

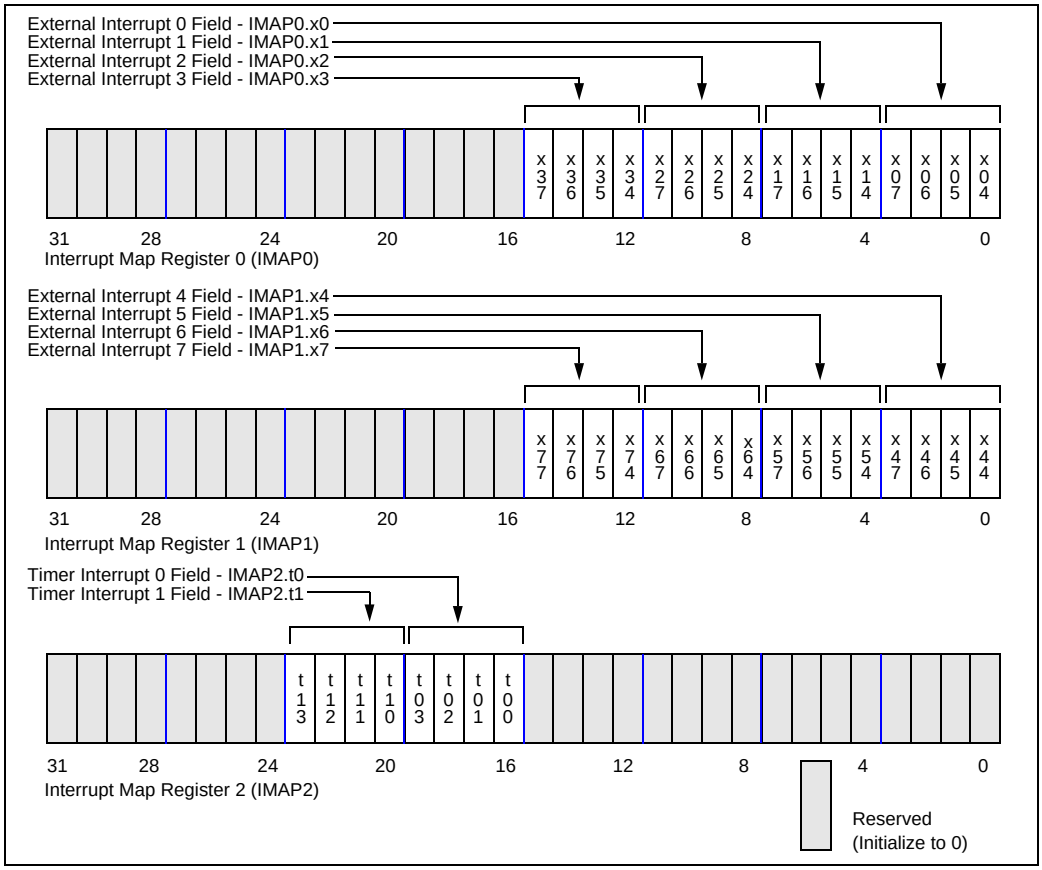

Figure 22. Interrupt Mapping Registers on the 80960Jx and 80960Hx Processors

The IMAPx registers are programmed by using the **sysctl** instruction. They may also be accessed as memory-mapped control registers, at addresses FF00.8520H through FF00.8528H, in supervisor-mode only.

6.3.2.3 Interrupt Mask and Interrupt Pending Registers (IMSK, IPND)

The interrupt mask register provides a mechanism to uniquely mask dedicated mode interrupts and timer interrupt sources, or globally mask expanded mode interrupts. In this manner, application software may prevent the servicing of these interrupts. The interrupt pending register serves as a means to post dedicated mode interrupts, or interrupts from a timer source. Application code which sets a bit in the interrupt pending register precipitates the same action that would occur if an external interrupt source caused the bit to be set.

These registers are accessed as special function registers: interrupt pending as sf0, interrupt mask as sf1. Note that the 80960Jx does not support special function registers. IMSK and IPND may be accessed as memory-mapped control registers, at address FF00.8504H and FF00.8500H, respectively. To access these memory-mapped registers, the application must be operating in supervisor mode. The **atmod**, **sysctl**, **ld**, and **st** instructions may also be used to modify these registers, again in supervisor-mode only.

On reset, interrupt mask is cleared (=0) and interrupt pending is undefined. Application code is responsible for explicitly initializing these registers. Figure 23 shows the format of these registers.

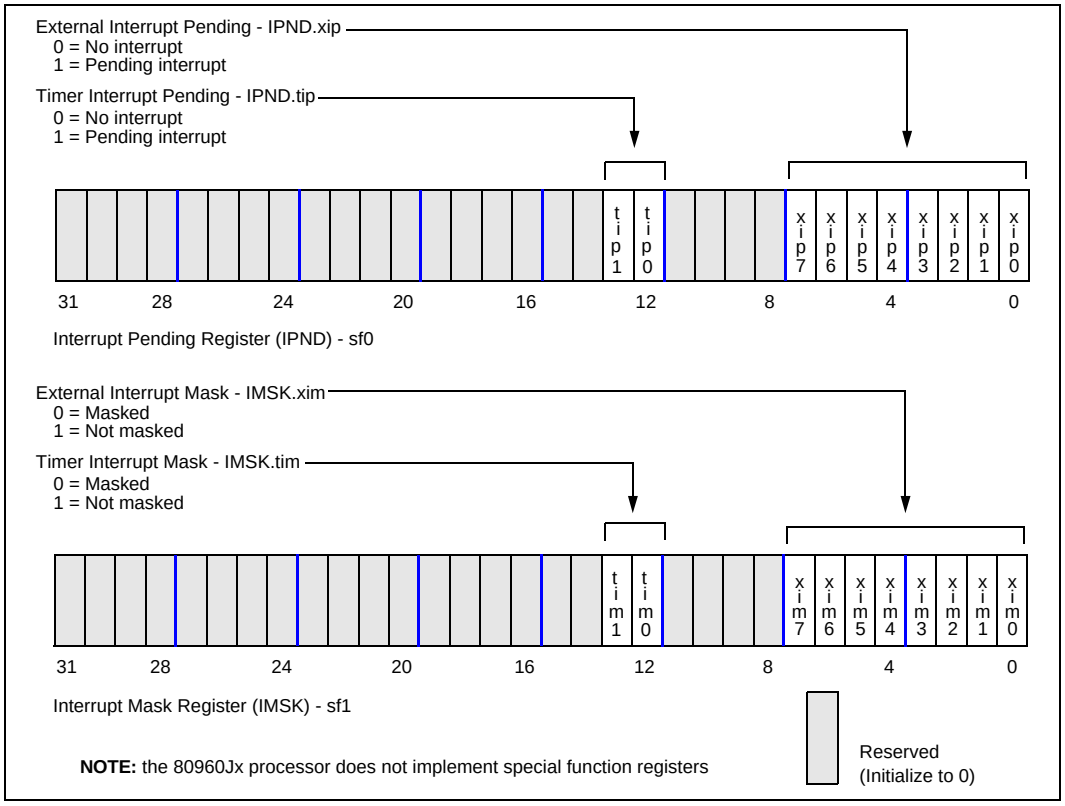

Figure 23. Interrupt Pending (sf0) and Interrupt Mask (sf1) on the 80960Jx and 80960Hx Processor

6.3.2.4 Improvements to Interrupt Latency

The 80960Jx and 80960Hx processors provide the ability to reserve local register frames in the on-chip register cache for use by interrupts of priority 28 or above. This mechanism minimizes the chances that high-priority interrupts result in a spill to external memory, potentially increasing interrupt latency. For more information on this topic, refer to Section 4.0, Register Cache/Stack Frames.

6.4 Timer Unit

The 80960Jx and 80960Hx processors integrate two general-purpose timers. Timers and timer controls are identical on the 80960Jx and 80960Hx processors. Refer to the *i960® Jx Microprocessor User's Manual* or the *i960®Hx Microprocessor User's Manual* for details. These 32-bit timers are useful as system clocks, or for other purposes. Each timer may be clocked independently (clock input based on CLKIN input). Timers operate in a single-shot mode, or auto-reload for continuous tick generation. Each timer may independently interrupt the core upon terminal count. Timer resources may be configured such that user-mode accesses generate a fault.

Timer resources are independently controlled by six control registers. These registers, along with their memory-mapped control register addresses, are listed in Table 17.

.

Table 17. Timer Control Registers on the 80960Jx and 80960Hx Processors

Name	Description	Memory-Mapped Address (Hex)
TRR0	Timer Reload Register 0	FF00.0300H
TCR0	Timer Count Register 0	FF00.0304H
TMR0	Timer Mode Register 0	FF00.0308H
TRR1	Timer Reload Register 1	FF00.0310H
TCR1	Timer Count Register 1	FF00.0314H
TMR1	Timer Mode Register 1	FF00.0318H

The timer mode control register enables/disables the timer, enables auto-reload, provides for protection from user-mode writes to timer resources, and controls the input clock to its associated timer. Timers may be clocked by:

- the bus clock (CLKIN)
- bus clock / 2
- bus clock / 4, or
- bus clock / 8

The timer count register contains the timer's current count, and may be read or written. The timer reload register contains the value to load into the timer count register upon reaching terminal count (when auto-reload mode is enabled).

The 80960Cx processor does not contain any integrated timers.

7.0 Memory-mapped Control Registers

The 80960Jx and 80960Hx processors implement memory-mapped control registers within the architecturally reserved memory locations from FF00.0000H to FFFF.FFFFH. Accesses to this address space **never** cause an external bus cycle. These memory-mapped control registers provide convenient access to control registers. Additionally, they provide visibility to internally cached values, such as the Processor Control Block (PRCB) pointer, which is not architecturally visible on the 80960Cx. **The 80960Cx processor *does not* implement memory-mapped control registers.**

Memory-mapped control registers residing in address locations FF00.0000H to FF00.7FFFH can be accessed in user or supervisor mode. Those located in addresses FF00.8000H to FFFF.FFFFH are accessible via supervisor mode only. When using load (**ld**) and store (**st**) instructions, application code must access memory-mapped registers as aligned words only. Unaligned and/or non-word accesses to these registers, or user mode accesses made to supervisor-only registers generate a fault. The **sysctl** instruction may also be used to atomically modify memory-mapped control registers. In addition, the **atmod** instruction may be used to atomically modify the Interrupt Pending (IPND) and Interrupt Mask (IMSK) registers.

Memory-mapped control register addresses are detailed in Section 8.0, Memory-mapped Control Register Address Space.

7.1 Special Function Registers and Memory-mapped Counterparts

The 80960Cx and 80960Hx processors implement special function registers (SFRs) to provide control over integrated peripherals. SFRs may be accessed by most REG format instructions. With the exception of the Cache Control Register, all of the 80960Hx SFRs are also accessible as memory-mapped control registers. **The 80960Jx *does not* implement special function registers.**

7.1.1 Special Function Registers on the 80960Cx Processor

The 80960Cx processor implements three special function registers:

- Interrupt Pending (sf0)
- Interrupt Mask (sf1), and
- DMA Command (sf2)

The formats of the Interrupt Pending and Interrupt Mask registers for the 80960Cx processor are shown in Section 6.3.1, 80960Cx Processor Interrupt Control Unit. DMA functionality is present only on the 80960Cx processor, and the DMA Command register as such is present only on the 80960Cx processor. The upper two bits (bits 30 and 31) of the DMA Command register control the data cache (CF processor only). For more information on DMA support, refer to the *i960® Cx Microprocessor User's Manual*.

7.1.2 Special Function Registers on the 80960Hx Processor

The 80960Hx processor implements five special function registers. With the exception of the data cache control register (sf2), these registers are also accessible as memory-mapped control registers. These registers, with their associated memory-mapped locations are listed in Table 18.

Table 18. Special Function Registers of the 80960Hx Processor

Name	Mnemonic	Special Function Register	Memory-mapped Address
Interrupt Pending Register	IPND	sf0	FF00.8500H
Interrupt Mask Register	IMSK	sf1	FF00.8504H
Data Cache Control Register	D	sf2	NA
Interrupt Control Register	INTCON	sf3	FF00.8510H
GMU Control Register		sf4	FF00.8000H

These registers may be accessed by application code as either special function or memory-mapped registers. However, these registers must not be accessed as both special function and memory-mapped registers within a short period of time (less than 64 core cycles). Doing so results in unpredictable behavior.

The formats of the interrupt pending, interrupt mask, and interrupt control registers are shown in Section 6.3.2, 80960Jx/80960Hx Processor Interrupt Control Unit. The format of the GMU Control Register is shown in Section 6.2, Guarded Memory Unit on the 80960Hx Processor, illustrates the format of the Data Cache Control Register.

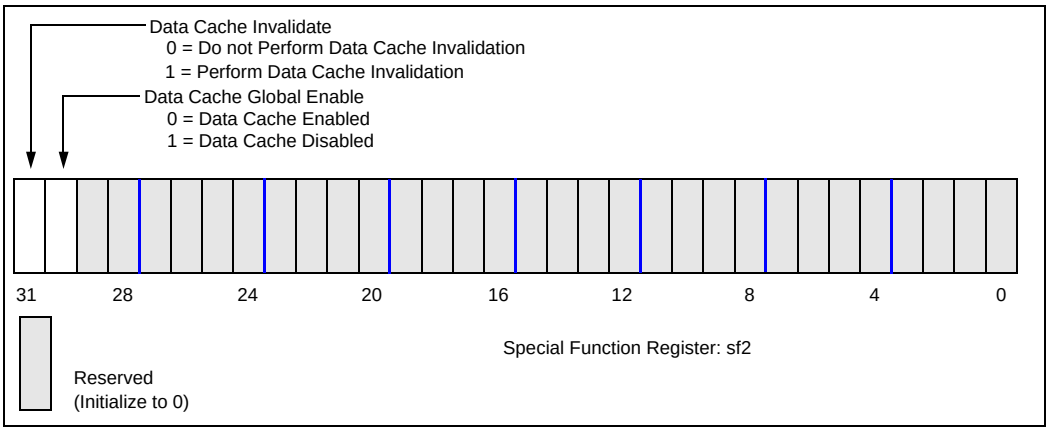

Figure 24. 80960Hx Processor Data Cache Control Register

The Data Cache Control register may be accessed as a special function register, or by use of the Data Cache Control instruction (**dcctl**). Regardless of how the register is accessed, bit 30 always properly reflects the state of the data cache: a zero if enabled and one if disabled. Writing a one to bit 31 causes the data cache to invalidate. Note that since the cache is write-through, it does not get flushed. When the invalidation process is complete, hardware automatically clears bit 31.

8.0 Memory-mapped Control Register Address Space

The memory-mapped control registers for the 80960Jx and 80960Hx microprocessors are shown in Table 19. Memory-mapped registers are not implemented on the 80960Cx processor. Accesses to this address space never causes an external bus cycle.

Table 19. Memory-mapped Control Registers (Sheet 1 of 5)

Address (Hex)	Description	Access Rights	Processor	
			Jx	Hx
FF00.0300H	Timer Reload Register 0 (TRR0)	S/U, R/W	X	X
FF00.0304H	Timer Count Register 0 (TCR0)	S/U, R/W	X	X
FF00.0308H	Timer Mode Register 0 (TMR0)	S/U, R/W	X	X
FF00.030CH	Reserved			
FF00.0310H	Timer Reload Register 1 (TRR1)	S/U, R/W	X	X
FF00.0314H	Timer Count Register 1 (TCR1)	S/U, R/W	X	X
FF00.0318H	Timer Mode Register 1 (TMR1)	S/U, R/W	X	X
FF00.031CH to FF00.7FFFH	Reserved			
FF00.8000H	GMU Control Register (GMCR, sf4)	S, R/W		X
FF00.8004H	Reserved			
FF00.8008H	Reserved			
FF00.8010H	Memory Protection Address 0 (MPAR0)	S, R/W		X
FF00.8014H	Memory Protection Mask 0 (MPMR0)	S, R/W		X
FF00.8018H	Memory Protection Address 1 (MPAR1)	S, R/W		X
FF00.801CH	Memory Protection Mask 1 (MPMR1)	S, R/W		X
FF00.8020H to FF00.807FH	Reserved			
FF00.8080H	Memory Detect Upper Bounds 0 (MDUB0)	S, R/W		X
FF00.8084H	Memory Detect Lower Bounds 0 (MDLB0)	S, R/W		X
FF00.8088H	Memory Detect Upper Bounds 1 (MDUB1)	S, R/W		X
FF00.808CH	Memory Detect Lower Bounds 1 (MDLB1)	S, R/W		X
FF00.8090H	Memory Detect Upper Bounds 2 (MDUB2)	S, R/W		X
FF00.8094H	Memory Detect Lower Bounds 2 (MDLB2)	S, R/W		X
FF00.8098H	Memory Detect Upper Bounds 3 (MDUB3)	S, R/W		X

Table 19. Memory-mapped Control Registers (Sheet 2 of 5)

Address (Hex)	Description	Access Rights	Processor	
			Jx	Hx
FF00.809CH	Memory Detect Lower Bounds 3 (MDLB3)	S, R/W		X
FF00.80A0H	Memory Detect Upper Bounds 4 (MDUB4)	S, R/W		X
FF00.80A4H	Memory Detect Lower Bounds 4 (MDLB4)	S, R/W		X
FF00.80A8H	Memory Detect Upper Bounds 5 (MDUB5)	S, R/W		X
FF00.80ACH	Memory Detect Lower Bounds 5 (MDLB5)	S, R/W		X
FF00.80B0H to FF00.80FFH	Reserved			
FF00.80E8H to FF00.80FFH	GMU Testability Registers			X
FF00.8100H	Default Logical Memory Configuration Register (DLMCON)	S, R/W	X	X
FF00.8104H	Reserved			
FF00.8108H	Logical Memory Address Register 0 (LMAR0)	S, R/W	X	X
FF00.810CH	Logical Memory Mask Register 0 (LMMR0)	S, R/W	X	X
FF00.8110H	Logical Memory Address Register 1 (LMAR1)	S, R/W	X	X
FF00.8114H	Logical Memory Mask Register 1 (LMMR1)	S, R/W	X	X
FF00.8118H	Logical Memory Address Register 2 (LMAR2)	S, R/W		X
FF00.811CH	Logical Memory Mask Register 2 (LMMR2)	S, R/W		X
FF00.8120H	Logical Memory Address Register 3 (LMAR3)	S, R/W		X
FF00.8124H	Logical Memory Mask Register 3 (LMMR3)	S, R/W		X
FF00.8128H	Logical Memory Address Register 4 (LMAR4)	S, R/W		X
FF00.812CH	Logical Memory Mask Register 4 (LMMR4)	S, R/W		X
FF00.8130H	Logical Memory Address Register 5 (LMAR5)	S, R/W		X
FF00.8134H	Logical Memory Mask Register 5 (LMMR5)	S, R/W		X
FF00.8138H	Logical Memory Address Register 6 (LMAR6)	S, R/W		X
FF00.813CH	Logical Memory Mask Register 6 (LMMR6)	S, R/W		X
FF00.8140H	Logical Memory Address Register 7 (LMAR7)	S, R/W		X
FF00.8144H	Logical Memory Mask Register 7 (LMMR7)	S, R/W		X
FF00.8148H	Logical Memory Address Register 8 (LMAR8)	S, R/W		X
FF00.814CH	Logical Memory Mask Register 8 (LMMR8)	S, R/W		X
FF00.8150H	Logical Memory Address Register 9 (LMAR9)	S, R/W		X
FF00.8154H	Logical Memory Mask Register 9 (LMMR9)	S, R/W		X
FF00.8158H	Logical Memory Address Register 10 (LMAR10)	S, R/W		X

Table 19. Memory-mapped Control Registers (Sheet 3 of 5)

Address (Hex)	Description	Access Rights	Processor	
			Jx	Hx
FF00.815CH	Logical Memory Mask Register 10 (LMMR10)	S, R/W		X
FF00.8160H	Logical Memory Address Register 11 (LMAR11)	S, R/W		X
FF00.8164H	Logical Memory Mask Register 11 (LMMR11)	S, R/W		X
FF00.8168H	Logical Memory Address Register 12 (LMAR12)	S, R/W		X
FF00.816CH	Logical Memory Mask Register 12 (LMMR12)	S, R/W		X
FF00.8170H	Logical Memory Address Register 13 (LMAR13)	S, R/W		X
FF00.8174H	Logical Memory Mask Register 13 (LMMR13)	S, R/W		X
FF00.8178H	Logical Memory Address Register 14 (LMAR14)	S, R/W		X
FF00.817CH	Logical Memory Mask Register 14 (LMMR14)	S, R/W		X
FF00.8180H to FF00.83FFH	Reserved			
FF00.8400H	Instruction Address Breakpoint 0 (IPB0)	S, Sys, WwG	X	X
FF00.8404H	Instruction address Breakpoint 1 (IPB1)	S, Sys, WwG	X	X
FF00.8408H	Instruction Address Breakpoint 2 (IPB2)	S, Sys, WwG		X
FF00.840CH	Instruction address Breakpoint 3 (IPB3)	S, Sys, WwG		X
FF00.8410H	Instruction Address Breakpoint 4 (IPB4)	S, Sys, WwG		X
FF00.8414H	Instruction address Breakpoint 5 (IPB5)	S, Sys, WwG		X
FF00.8418H to FF00.841FH	Reserved			
FF00.8420H	Data Address Breakpoint 0 (DAB0)	S, R/W, WwG	X	X
FF00.8424H	Data Address Breakpoint 1 (DAB1)	S, R/W, WwG	X	X
FF00.8428H	Data Address Breakpoint 2 (DAB2)	S, R/W, WwG		X
FF00.842CH	Data Address Breakpoint 3 (DAB3)	S, R/W, WwG		X
FF00.8430H	Data Address Breakpoint 4 (DAB4)	S, R/W, WwG		X
FF00.8434H	Data Address Breakpoint 5 (DAB5)	S, R/W, WwG		X
FF00.8438H to FF00.843FH	Reserved			
FF00.8440H	Breakpoint Control (BPCON)	S, R/W, WwG	X	X
FF00.8444H	Expanded Breakpoint Control (XBPCON)	S, R/W, WwG		X
FF00.8448H to FF00.84FFH	Reserved			
FF00.8500H	Interrupt Pending (IPND, sf0)	S, R/W, AtMod	X	X

Table 19. Memory-mapped Control Registers (Sheet 4 of 5)

Address (Hex)	Description	Access Rights	Processor	
			Jx	Hx
FF00.8504H	Interrupt Mask (IMSK, sf1)	S, R/W, AtMod	X	X
FF00.8508H to FF00.850FH	Reserved			
FF00.8510H	Interrupt Control (ICON, sf3)	S, R/W	X	X
FF00.8514H to FF00.851FH	Reserved			
FF00.8520H	Interrupt Map 0 (IMAP0)	S, R/W	X	X
FF00.8524H	Interrupt Map 1 (IMAP1)	S, R/W	X	X
FF00.8528H	Interrupt Map 2 (IMAP2)	S, R/W	X	X
FF00.852CH to FF0085FFH	Reserved			
FF00.8600H	80960Jx: Physical Memory Control Region 0:1 (PMCON0:1) 80960Hx: Physical Memory Control Region 0 (PMCON0)	S, R/W	X	X
FF00.8604H	Physical Memory Control Region 1 (PMCON1)	S, R/W		X
FF00.8608H	80960Jx: Physical Memory Control Region 2:3 (PMCON2:3) 80960Hx: Physical Memory Control Region 2 (PMCON2)	S, R/W	X	X
FF00.860CH	Physical Memory Control Region 3 (PMCON3)	S, R/W		X
FF00.8610H	80960Jx: Physical Memory Control Region 4:5 (PMCON4:5) 80960Hx: Physical Memory Control Region 4 (PMCON4)	S, R/W	X	X
FF00.8614H	Physical Memory Control Region 5 (PMCON5)	S, R/W		X
FF00.8618H	80960Jx: Physical Memory Control Region 6:7 (PMCON6:7) 80960Hx: Physical Memory Control Region 6 (PMCON6)	S, R/W	X	X
FF00.861CH	Physical Memory Control Region 7 (PMCON7)	S, R/W		X
FF00.8620H	80960Jx: Physical Memory Control Region 8:9 (PMCON8:9) 80960Hx: Physical Memory Control Region 8 (PMCON8)	S, R/W	X	X
FF00.8624H	Physical Memory Control Region 9 (PMCON9)	S, R/W		X
FF00.8628H	80960Jx: Physical Memory Control Region 10:11 (PMCON10:11) 80960Hx: Physical Memory Control Region 10 (PMCON10)	S, R/W	X	X

Table 19. Memory-mapped Control Registers (Sheet 5 of 5)

Address (Hex)	Description	Access Rights	Processor	
			Jx	Hx
FF00.862CH	Physical Memory Control Region 11 (PMCON11)	S, R/W		X
FF00.8630H	80960Jx: Physical Memory Control Region 12:13 (PMCON12:13) 80960Hx: Physical Memory Control Region 12 (PMCON12)	S, R/W	X	X
FF00.8634H	Physical Memory Control Region 13 (PMCON13)	S, R/W		X
FF00.8638H	80960Jx: Physical Memory Control Region 14:15 (PMCON14:15) 80960Hx: Physical Memory Control Region 14 (PMCON14)	S, R/W	X	X
FF00.863CH	Physical Memory Control Region 15 (PMCON15)	S, R/W		X
FF00.8640H to FF00.86FBH	Reserved			
FF00.86FCH	Bus Configuration Control (BCON)	S, R/W	X	X
FF00.8700H	Processor Control Block Pointer (PRCBptr)	S, RO	X	X
FF00.8704H	Interrupt Stack Pointer (ISP)	S, R/W	X	X
FF00.8708H	Supervisor Stack Pointer (SSP)	S, R/W	X	X
FF00.870CH	Reserved			
FF00.8710H	Device Identification (DEVICEID)	RO	X	X
FF00.8714H to FFFF.FFFFH	Reserved			

NOTES:

1. AtMod: Register can be updated quickly through the **atmod** instruction.
   RO: Read only.
   R/W: Register readable or writable.
   S: Supervisor access only.
   Spcl: Special Intel only test register.
   S/U: Supervisor or User access allowed, may be configured to be supervisor only.
   Sys: Modifiable using the **sysctl** instruction.
   WwG: Writing or modifying (with **st** or **sysctl**) the register only allowed when access has been granted.

Register locations which are not implemented are specifically reserved by Intel. Any locations not specifically defined are reserved by Intel.

Note that access to the instruction and data address breakpoint registers requires that the processor grant permission to access these registers. This is discussed in further detail in Section 5.0, Breakpoint Resource Sharing Mechanism.

9.0 80960Hx/80960Cx Pin Compatibility

Like the 80960Cx, the 80960Hx is available in the 168-lead PGA package. However, the 80960Hx is not 100% pin-compatible with the 80960Cx. This is largely due to the lack of Direct Memory Access (DMA) capability on the 80960Hx. On the 80960Cx, DMA capability consumed thirteen pins. The 80960Hx incorporates enhanced bus control functions and JTAG; pins required to support these features replace the DMA pins present on the 80960Cx. In this manner, it is possible to design a board that accepts the 80960Cx processor now, and the 80960Hx in the future. Pin diagrams for the 168-lead PGA package are illustrated in Figure 25 and Figure 26. See *Application Note 506, Designing for 80960Cx and 80960Hx Compatibility* for more details.

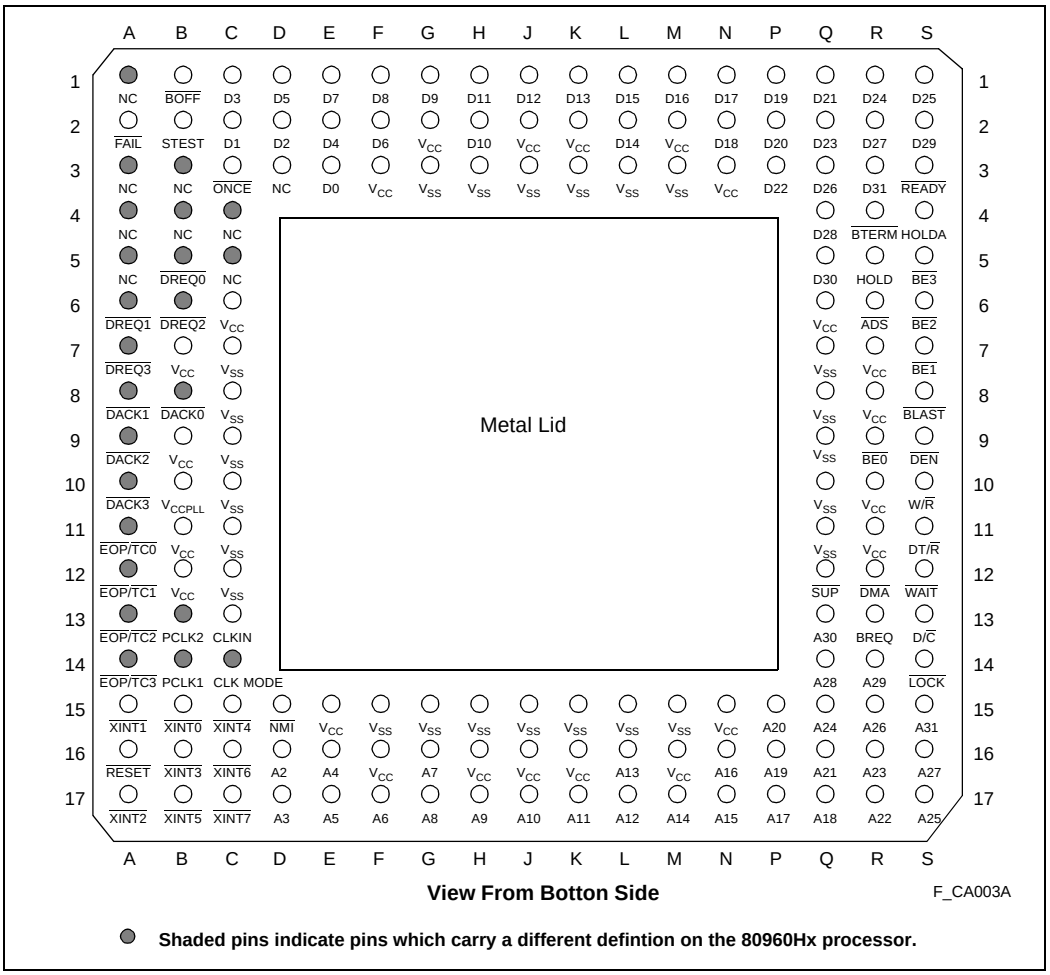

Figure 25. PGA Pinout Diagram for the 80960Cx Processor

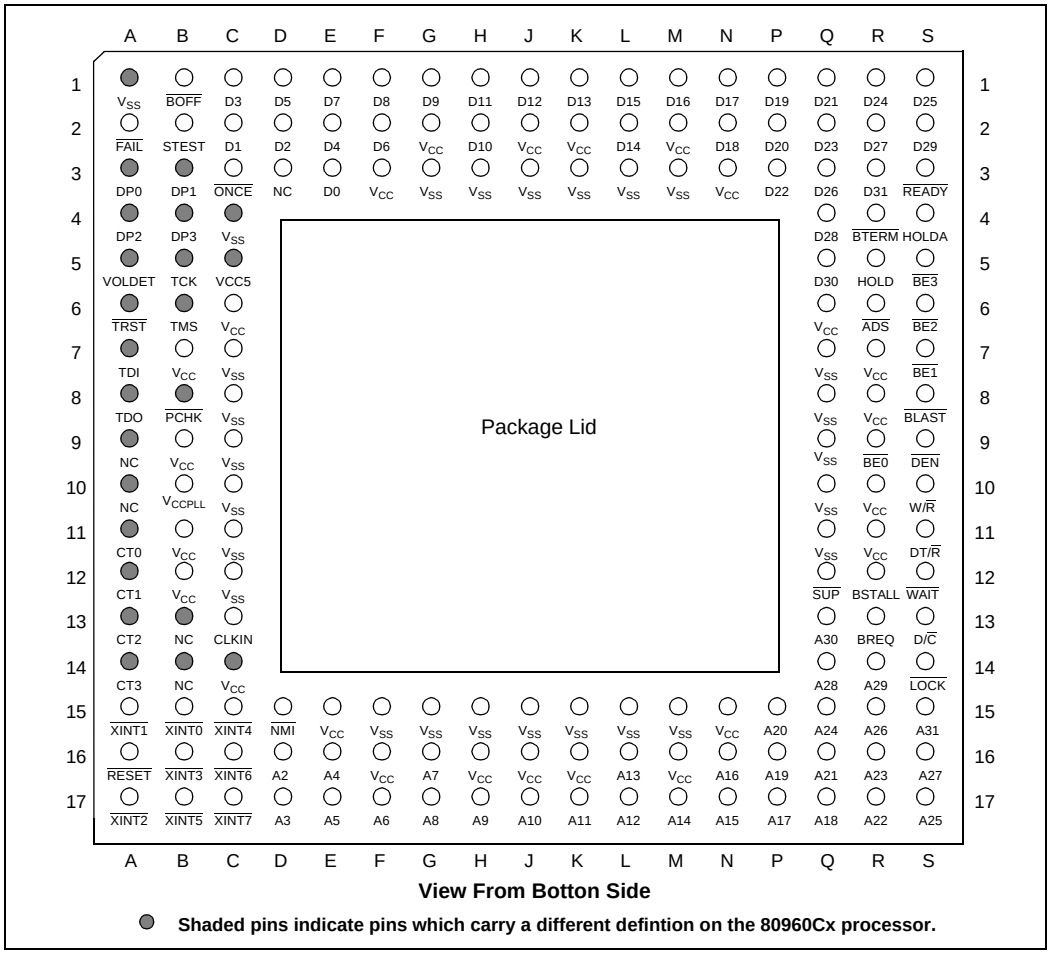

Figure 26. PGA Pinout Diagram for the 80960Hx Processor

The 80960Jx incorporates a multiplexed address and data bus. Therefore, pin compatibility with the 80960Cx or 80960Hx is not required or relevant.

10.0 Conclusion

In conclusion, this document describes three implementations of the i960® architecture: the 80960Cx, 80960Jx, and 80960Hx microprocessors.

During the 80960Jx and 80960Hx definition process, every attempt was made to maintain backward compatibility with the 80960Cx processor. To a large extent, this compatibility is maintained; most application code will run on each processor without modification. Due to enhancements to the bus interface, instruction set, and other refinements, complete compatibility was not maintained. In summary, this document discusses in detail these differences between the 80960Cx, 80960Jx, and 80960Hx microprocessors.

11.0 Related Information

This appnote contains condensed information from the *i960® Cx Microprocessor User's Manual*, *Application Note 506, Designing for 80960Cx and 80960Hx Compatibility*, the *i960® Jx Microprocessor User's Manual*, and the *i960® Hx Microprocessor User's Manual*. The reader should refer to these documents for more detailed information. The following table shows these document's order numbers:

Document Name	Order #
i960® Cx Microprocessor User's Manual	270710
i960® Jx Microprocessor User's Manual	272483
i960® Hx Microprocessor User's Manual	272484
Application Note 506—Designing for 80960Cx and 80960Hx Compatibility	272556

To receive these documents or any other available Intel literature, contact:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect IL 60056-7641
1-800-879-4683