80960JA/JF/JD
SPECIFICATION UPDATE

Release Date: July, 1996

Order Number: 272852-001

The 80960JA/JF/JD may contain design defects or errors known as errata. Characterized errata that may cause the 80960JA/JF/JD's behavior to deviate from published specifications are documented in this specification update.

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

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

or call in North America 1-800-879-4683, Europe 44-0-1793-431-155, France 44-0-1793-421-777,

Germany 44-0-1793-421-333 other Countries 708-296-9333

CONTENTS

REVISION HISTORY

The 80960JA/JF/JD series was introduced on June 6, 1994. The A-0 stepping was the initial silicon release. The 80960Jx family is now in production on the A-2 stepping.

Date of Revision	Version	Description
07/01/96	001	This is the new Specification Update document. It contains all identified errata published on or before this date.
11/29/95	2.2	Added new errata #16: VIH level on TRST#/RESET# Greater than Specified in Data Sheets.
10/31/95	2.1	Added new errata #15: Actual Max Tov Greater than Specified in Data Sheets.
3/30/95	2.0	Added the 80960JD to the errata sheet. Added the A-2 stepping, whose only difference from the A-0 stepping is the fix of the RDYRCV# errata during a Th(hold cycle). A new technote was added to the list of 80960Jx technotes.
3/10/95	1.9	Added new errata section User's Manual Errata containing only application critical users's manual errata. Added new user's manual errata #1: Manual Errata: IPND should not be Modified using a MEM Format Instruction. Two lines of code in errata #4 Fault Stack Alignment were changed as shown: from: sele r5, r10, r8 to: sele r5, r10, r4 from: sele r5, r10, r9 to: sele r5, r10, r5
2/9/95	1.8	Added new errata #13: Data Breakpoints on System Procedure Entries are Lost for Certain Fault Types**.**
1/23/95	1.7	Added new errata #12: "balx" Instruction Does Not Branch When targ and dst Use the Same Register.

PREFACE

As of July, 1996, Intel's Semiconductor Products Group has consolidated available historical device and documentation errata into this new document type called the Specification Update. We have endeavored to include all documented errata in the consolidation process, however, we make no representations or warranties concerning the completeness of the Specification Update.

This document is an update to the specifications contained in the Affected Documents/Related Documents table below. This is the first release of the Specification Update. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Information types defined in Nomenclature are consolidated into the specification update and are no longer published in other documents.

This document may also contain additional information that was not previously published.

Affected Documents/Related Documents

Title	Order
i960® Jx Microprocessor User's Manual	272483-001
80960JA/JF Embedded 32-Bit Microprocessor data sheet	272504-004
80L960JA/JF 3.3 V Embedded 32-Bit Microprocessor data sheet	272744-002
80960JD Embedded 32-Bit Microprocessor data sheet	272596-002
AP- 712: DRAM Controller for i960® JA/JF/JD Processors	272674-001
AP-716: Architectural Comparison 80960Cx/Jx/Hx	272694-001
AP-727: Interfacing the i960® Jx Processor to NEC SAR	272779-001

Nomenclature

Errata are design defects or errors. These may cause the published (component, board, system) behavior to deviate from published specifications. Hardware and software designed to be used with any component, board, and system must consider all errata documented.

Specification Changes are modifications to the current published specifications. These changes will be incorporated in any new release of the specification.

Specification Clarifications describe a specification in greater detail or further highlight a specification’s impact to a complex design situation. These clarifications will be incorporated in any new release of the specification.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in any new release of the specification.

NOTE:

Errata remain in the specification update throughout the product’s lifecycle, or until a particular stepping is no longer commercially available. Under these circumstances, errata removed from the specification update are archived and available upon request. Specification changes, specification clarifications and documentation changes are removed from the specification update when the appropriate changes are made to the appropriate product specification or user documentation (datasheets, manuals, etc.).

SUMMARY TABLE OF CHANGES

The following table indicates the errata, specification changes, specification clarifications, and documentation changes which apply the 80960JA/JF/JD product. Intel may fix some of the errata in a future stepping of the component, and account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

Codes Used in Summary Table

Steps

X:	Errata exists in the stepping indicated. Specification Change or Clarification that applies to this stepping.
(No mark) or (Blank box):	This erratum is fixed in listed stepping or specification change does not apply to listed stepping.

Page

(Page):	Page location of item in this document.

Status

Doc:	Document change or update will be implemented.
Fix:	This erratum is intended to be fixed in a future step of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Eval:	Plans to fix this erratum are under evaluation.

Row

|	Change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Errata

No.	Steppings		Page	Status	ERRATA
	A-0	A-2			
9600001	X		8	Fixed	RDYRCV# Restriction During Ta, Th and Ti Cycles (A-0 Stepping Only)
9600002	X	X	9	Fix	RDYRCV# Restriction During Ta and Ti Cycles (A-2 Stepping Only)
9600003	X	X	10	Fix	System-Local Fault Calls Use System-Supervisor Trace Enable Bit
9600004	X	X	10	Fix	Instructions “inten” and “intdis” Not Fully Implemented
9600005	X	X	10	Fix	Fault Stack Alignment
9600006	X	X	13	Fix	Software Interrupt Erratum
9600007	X	X	14	Fix	Pullup on LOCK#/ONCE# Pin Does Not Turn Off
9600008	X	X	14	Fix	Software Reinitialization Values in LMMR0, LMMR1, DLMCON
9600009	X	X	14	Doc	Power Supply Current (Icc) Higher than Anticipated
9600010	X	X	15	Fix	One Cycle Performance Hit Due to Instruction Order with LOAD Instructions
9600011	X	X	15	Fix	“divi” Instruction Performance Hit When src2 = dst
9600012	X	X	15	Fix	Instructions Executed Between Back to Back Interrupts
9600013	X	X	16	Fix	“balx” Instruction Does Not Branch When targ and dst Use the Same Register
9600014	X	X	16	Fix	Data Breakpoints on System Procedure Entries are Lost for Certain Fault Types
9600015	X	X	17	Fix	Actual Max Tov Greater than Specified in Data Sheets
96000016	X	X	17	Fix	VIH level on TRST#/RESET# Greater than Specified in Data Sheets

Specification Changes

No.	Steppings			Page	Status	SPECIFICATION CHANGES
	#	#	#			
						None for this revision of this specification update.

Specification Clarifications

No.	Steppings			Page	Status	SPECIFICATION CLARIFICATIONS
	#	#	#			
						None for this revision of this specification update.

Documentation Changes

No.	Document Revision	Page	Status	DOCUMENTATION CHANGES
001	272483-001	19	Fix	IPND should not be modified using a MEM Format Instruction
002	272483-001	19	Fix	Page 7-9
003	272483-001	19	Fix	Page 9-10
004	272483-001	19	Fix	Page 9-3, Table 9-1
005	272483-001	20	Fix	Page 12-10, section 12.6.1
006	272483-001	20	Fix	Page C-23, Figure C-25
007	272483-001	20	Fix	Page 11-29, Example 11-6
008	272483-001	20	Fix	Page 17-09, Section 17. 3.5
009	272483-001	21	Fix	Page 13-08, Section 13.2.6.3
010	272483-001	21	Fix	Page 9-8, Table 9-3
011	272483-001	21	Fix	Page 14-10, Section 14.3
012	272483-001	22	Fix	Page 6-57, Section 6.6.32
013	272483-001	22	Fix	Page 11-13, Example 11-1
014	272483-001	22	Fix	Page 3-9, Table 3-4
015	272483-001	23	Fix	Page 11-13, Example 11-1
016	272483-001	23	Fix	Page 3-22
017	272483-001	23	Fix	Pages B-1, B-6, and B7
018	272483-001	23	Fix	Page 13-21, Table 13-10
019	272483-001	24	Fix	Page 2-5, Table 2-4

IDENTIFICATION INFORMATION

Markings

80960JA/JF/JD processors may be identified electrically according to device type and stepping.

Stepping register

The following table lists the devices to which this errata sheet applies:

Device and Stepping	Identifier in g0
80960JA A0 & A2	0x08821013
80960JD A0 & A2	0x08820013
80960JF A0 & A2	0x08820013

Refer to the data sheet for instructions on how to obtain the identifier number.

ERRATA

9600001. RDYRCV# Restriction During Ta, Th and Ti Cycles (A-0 Stepping Only)

PROBLEM: The RDYRCV# pin indicates that data on the address-data (AD) lines can be sampled or removed. If RDYRCV# is not asserted during a Td cycle, the Td cycle is extended to the next cycle by inserting a wait state (Tw).

Normally the processor ignores this pin during the address state (Ta). On this stepping, however, the processor can recognize the assertion of RDYRCV# during the address state of an access and prematurely terminate the access. For example, in a quad-word load (with RDYRCV# asserted during Ta) the processor will erroneously take the first word of data off the bus during the address cycle, and will then read only three more words of data. This will cause the last word of data to be lost, and the first three will be corrupted. The processor also incorrectly responds to RDYRCV# assertion during the idle (Ti) and (Th) states. Under these conditions, the processor will deassert HOLDA (if asserted), one clock after the RDYRCV# assertion. Hold acknowledge can never be asserted (or reasserted) until one clock after RDYRCV# is sampled high.

IMPLICATION: The RDYRCV# generation logic must be designed to only assert RDYRCV# during either the Td or Tr states, and should not be used by a multi-master system for any of the other bus masters.

WORKAROUND: Make sure that the system implementation does not allow RDYRCV# assertion during any state except valid data (Td) and recovery (Tr) states. Typical synchronous state logic for the 80960Jx can be described as “normally-not-ready” (i.e., the pin is only asserted when the system is ready). Normally-not-ready logic will usually satisfy the requirement without modification.

Strict “normally-ready” state logic cannot be used in 80960Jx systems because the logical sense of RDYRCV# changes during the recovery state. Driving RDYRCV# low indefinitely during Tr would, by definition, cause the processor to hang in Tr states indefinitely. If the ready logic resembles a normally-ready system, change it to normally-not-ready.

If the 80960Jx is the sole bus master, multiple open-drain ready signals can be wire-OR’ed with a pull-up resistor to drive RDYRCV#. Be sure to satisfy the processor’s setup and hold timing requirements at the end of every bus clock.

If the 80960Jx is part of a multi-master system, do not share the RDYRCV# signal with the other bus masters. Providing separate ready signals will prevent the 80960Jx from spuriously deasserting HOLDA.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600002. RDYRCV# Restriction During Ta and Ti Cycles (A-2 Stepping Only)

PROBLEM: The RDYRCV# signal indicates that data on the AD lines can be sampled or removed. If RDYRCV# is not asserted during a Td cycle, the Td cycle is extended to the next cycle by inserting a wait state (Tw).

Normally the processor ignores this pin during the address state (Ta). On this stepping, however, the processor can recognize the assertion of RDYRCV# during the address state of an access and prematurely terminate the access. For example, in a quad-word load (with RDYRCV# asserted during Ta) the processor erroneously takes the first word of data off the bus during the address cycle, and then reads only three more words of data. This causes the last word of data to be lost, and the first three are corrupted.

The processor also incorrectly responds to RDYRCV# assertion during the idle (Ti) state.

IMPLICATION: The RDYRCV# generation logic must be designed to only assert RDYRCV# during either the Td or Tr states.

Typical synchronous state logic for the 80960Jx can be described as “normally-not-ready” (i.e., the pin is only asserted when the system is ready). Normally-not-ready logic will usually satisfy the requirement without modification.

WORKAROUND: Make sure that the system implementation does not allow RDYRCV# to be asserted during any state except valid data (Td) and recovery (Tr) states. Typical synchronous state logic for the 80960Jx can be described as “normally-not-ready” (i.e., the pin is only asserted when the system is ready). Normally-not-ready logic will usually satisfy the requirement without modification.

Strict “normally-ready” state logic cannot be used in 80960Jx systems because the logical sense of RDYRCV# changes during the recovery state. Driving RDYRCV# low indefinitely during Tr would, by definition, cause the processor to hang in Tr states indefinitely. If the ready logic resembles a normally-ready system, change it to normally-not-ready.

If the 80960Jx is the sole bus master, multiple open-drain ready signals can be wire-OR’ed with a pull-up resistor to drive RDYRCV#. Be sure to satisfy the processor’s setup and hold timing requirements at the end of every bus clock.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600003. System-Local Fault Calls Use System-Supervisor Trace Enable Bit

PROBLEM: When a fault handler is implemented through a system-local fault call, the fault handler IP is found in the system procedure table. All other characteristics of the call are those of a local call: PC.te (trace enable bit in the Process Controls), PC.em (execution mode flag in the Process Controls) remain unchanged and there is no stack switch. A system-local fault call on the 80960Jx has all of the previously mentioned characteristics, except that PC.te is copied from SSP.te (trace control bit in the Supervisor Stack Pointer). This erratum could cause tracing to be toggled when entering a system-local fault call. In addition when the processor is in user mode while executing the fault handler, PC.te will not be restored upon the return from the fault handler.

IMPLICATION: Little to none. Use system-supervisor calls or a local-call handler instead of a system-local fault handler.

WORKAROUND: Do not use the system-local fault handlers. If a local fault handler is required, use a local-call handler and place the handler pointer directly in the fault table. The other option is to use only system-supervisor calls from the system procedure table. If system-local fault handlers must be used, then the designer should make sure that the PC.te in user mode is the same as SSP.te in the System Procedure Table.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600004. Instructions "inten" and "intdis" Not Fully Implemented

PROBLEM: Two new instructions used for interrupt control, **inten** and **intdis**, on the 80960Jx were not fully implemented.

IMPLICATION: Must use **intctl** instruction instead of either **inten** or **intdis**; however, **intctl** takes more cycles to execute.

WORKAROUND: Do not use **inten** and **intdis** for globally enabling and disabling interrupts. Instead, use the interrupt control instruction **intctl** which can both enable and disable interrupts.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600005. Fault Stack Alignment

PROBLEM: In some situations, when a fault occurs, an unaligned (not quad-word aligned) fault record is written to the stack; as a result, a return-from-supervisor-mode-fault incorrectly restores the arithmetic controls (AC) register and process controls (PC) register. Furthermore, while within the fault handler, the FP incorrectly points to the beginning of the fault record. This case happens when:

(1) a local fault handler is selected for the fault or the current execution mode is Supervisor when the fault occurs; and

(2) the current stack is offset from a quad-word boundary by 1 to 8 bytes:

SP = 16 x N + 1
SP = 16 x N + 2
SP = 16 x N + 3
SP = 16 x N + 4
SP = 16 x N + 5
SP = 16 x N + 6
SP = 16 x N + 7
SP = 16 x N + 8

Where N is a positive integer.

The other possible values of SP do not generate the erratum:
SP = 16 x N and SP = 16 x N + 9 through SP = 16 x N + 15

Code Sequences Affected

Assuming the SP described above, the erratum manifests itself any time a fault is taken and does not result in a stack switch (all cases except system supervisor fault taken from user mode). This forces the fault record to become misaligned with respect to the FP and the stack pointer (SP) to become non-quad-word aligned. The fault type and the fault IP fields of the fault record are then stored at address FP and FP+4 respectively. The user cannot retrieve information from the fault record at the location expected in memory. Upon return from the fault handler the AC is cleared. The PC is also cleared upon return if the fault handler was executing in supervisor mode.

IMPLICATION: Requires extra code in every fault handler and extra time to implement the workaround.

WORKAROUND: The workaround requires special code at the beginning of each fault handler that corrects the unaligned fault record and makes accessing the fault record and returning from the fault handler operate correctly. There is one limitation to the workaround (discussed in further detail below). The workaround code is guaranteed to work on all versions of the 80960Jx present and future. The workaround consists of assembly code that rewrites the entire fault record to an aligned location when the situation is detected. This code should be placed at the beginning of all fault handlers.

Restriction: If the application is composed only of C code compiled by the Intel CTools or GNU C compiler, this anomaly cannot occur. If an application is compiled by another C compiler or contains customer routines coded directly in 80960 assembly code, it may be possible for the improper fault record to be generated when a fault occurs. For code generated by the latter method, a code restriction should be followed to ensure that the improper fault record is not generated: if the SP is always quadword aligned, (i.e., if SP is always incremented and decremented by multiples of 16) the proposed workaround is not needed.

```
## Fix Start - Place at beginning of all fault handlers.
## Uses r3 through r15
and not      0xf, sp, r3
cmpobe sp, r3, 1f   ## No fix needed if equal

## Problem detected: Fix Unaligned Fault Record
lda 24(sp),sp       ## Fix the stack to be quadword aligned
lda 32(fp),fp       ## Fix the frame pointer
lda -72(fp), r3     ## r3 = base for fault record
not 0, r5           ## r5 = 0xFFFF FFFF Wrong type of value
ldl 40(r3), r10     ## r10= 1st fault type (from fault record)
                    ## r11= 1st fault IP (from fault record)
subo 24, sp, r12    ## r12= original value of SP
## Were the fault type and fault IP clobbered by a frame spill?
cmpo r11, r12       ## is the SP = first fault IP?
                    ## <=> has an interrupt occurred?
                    ##     No | Yes
sele r5, r10, r4    ## r4 = fault type* | 0xFFFF FFFF
sele r5, r11, r5    ## r5 = fault IP* | 0xFFFF FFFF
                    ## * = from original fault record
## Get information from the existing fault record
ldq -24(r3),r8      ## Get 2nd Fault type, Fault Address,
                    ## Get saved TC, fault IP (resumption record)
ldq 24(r3),r12      ## Get Fault Data, Otype, saved PC, AC
## Write new fault record
stl r4, 64(r3)      ## Store Fault Type & Fault IP
stl r8, (r3)        ## Write 2nd Fault Address and Fault Type
stl r10, 8(r3)      ## Write Record
stl r12, 48(r3)     ## Write Fault Data and Otype
stl r14, 56(r3)     ## Write Saved PC, AC
## Fix End
1:                  ## Start of actual fault handler code.
```

Limitations of the Workaround

The workaround will not work if the following conditions occur:

- An interrupt occurs on the way to the fault handler.
- The local register frame of the fault handler is spilled due to the interrupt.

When this limitation occurs and the fault record is incomplete, the value 0xFFFF FFFF is written in the fault type and fault IP fields of the new fault record (created by the workaround code listed above).

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600006. Software Interrupt Erratum

PROBLEM: When the processor is executing in supervisor mode at process priority N due to a fault or interrupt that comes from user mode at a lower process priority, a software interrupt is posted at priority S, where $S \leq N$. Then a fault or interrupt return to user mode is executed. When the PC is restored due to the return to user mode the process priority is reset to U, where $U<S$. As a result, before the first instruction in user mode can execute, control is transferred to the software interrupt handling mechanism for the interrupt posted at priority S.

The error occurs when the software interrupt handling mechanism updates the interrupt table and Software-Interrupt Priority Register *before* switching to Supervisor mode. Because the processor is in the wrong mode, a type.mismatch fault will occur and control will be transferred to the fault handling mechanism before the first instruction of the software interrupt handler can be issued. Upon return from the fault handler, execution of the interrupt handler will continue; however, subsequent software interrupts may be lost.

IMPLICATION: Overhead is needed to make sure software interrupts do not occur in user mode, or the loss of software interrupt capabilities will result.

WORKAROUND: The only workarounds for this erratum are either not allowing software interrupts while the processor is in user mode or avoiding the use of software interrupts altogether.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600007. Pullup on LOCK#/ONCE# Pin Does Not Turn Off

PROBLEM: The pin for the LOCK#/ONCE# signals is provided with a weak pullup device intended to keep the processor from accidentally entering ONCE mode at reset. The pullup device is supposed to turn off after the deassertion of RESET#. However, the pullup remains on, even in the ONCE high impedance mode.

The strength of the pullup device was measured at 140 µA, which means it looks approximately like a 35k resistor attached to Vcc.

IMPLICATION: Ensure driver strengths will overdrive the pullup transistor.

WORKAROUND: No workaround is necessary; the erratum mostly affects factory test procedures. The LOCK# pulldown transistor and any reasonable driver attached to the pin during ONCE mode have ample strengths to overdrive the pullup transistor.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600008. Software Reinitialization Values in LMMR0, LMMR1, DLMCON

PROBLEM: After software reinitialization, LMMR0.lmte, LMMR1.lmte, and DLMCON.dcen should be zero (refer to the *i960Jx Microprocessor User's Manual*, Table 11-2). In the current implementation, of both the processor and the user's manual, these bits retain their values prior to the software reinitialization.

IMPLICATION: Software should take into account the fact that these values may change on a future stepping of the 80960JX and should be able to handle both current and future values of these registers.

WORKAROUND: None.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600009. Power Supply Current (Icc) Higher than Anticipated

PROBLEM: The Icc values measured on early silicon samples are approximately 10 - 20% higher than the targeted values in the data sheet.

IMPLICATION: User must account for higher current required from the power supply.

WORKAROUND: Intel will publish corrected Icc when device is fully characterized.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600010. One Cycle Performance Hit Due to Instruction Order with LOAD Instructions

PROBLEM: Under certain conditions, a LOAD instruction immediately followed by a REG instruction could cause an extra bus cycle to be added to the execution time. This is due to internal processor bus availability and is non-predictable.

IMPLICATION: Instructions may need to be re-ordered to avoid performance loss.

WORKAROUND: For critical code sequences that contain LOAD's followed by a REG instruction, experiment with the order of the instructions that follow the LOAD instruction. This may eliminate the extra cycle. There is no workaround for this problem.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600011. "divi" Instruction Performance Hit When src2 = dst

PROBLEM: Execution of the **divi** instruction takes approximately 40 extra cycles when the register used in the src2 operand is equal to the dst operand register.

IMPLICATION: One less register is available if performance hit is to be avoided.

WORKAROUND: To avoid the ~40 cycle performance hit for **divi**, do not use the same register for src2 and dst. If the application is composed only of C code compiled by the Intel CTools or GNU C compiler version 4.6 or above, then this situation will not occur.

STATUS: Intel plans to fix on subsequent steppings of the 80960JA/JF/JD.

9600012. Instructions Executed Between Back to Back Interrupts

PROBLEM: The processor can insert instructions from interrupted process code in between back to back interrupts under certain conditions. This situation could cause a slightly longer interrupt latency depending on the instruction(s) executed.

This situation occurs when the processor is executing an interrupt handler, and an interrupt(s) is pending at a priority which is lower than the current interrupt handler priority and higher than the interrupted process priority. Upon return from the current interrupt handler, the processor retains the interrupt handler process priority for 4-5 cycles. This prevents lower priority interrupts from being serviced immediately and allows interrupted process code to be executed until the priority is lowered. Extra cycles will be added to the interrupt latency of the pending interrupts based on the instructions executed during the process priority interim.

IMPLICATION: Increased interrupt latency may result.

WORKAROUND: There is no workaround for this latency problem.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600013. "balx" Instruction Does Not Branch When targ and dst Use the Same Register

PROBLEM: When the targ and dst operands of **balx** use the same register, **balx** does not branch. Typically, the targ register holds the address to branch to, specified by the user, and the dst register is automatically loaded by the processor with the address to return to. When the same register is used for the dst and the targ, the targ register, after being used to calculate the target address, receives the processor-loaded address to return to. Architecturally, using the same registers for targ and dst is permissible and works on other 80960 family members, but on the 80960Jx this causes the **balx** to become non-functional.

In the following example, balx should compute a target address of "xyz", load "abc" into r12, and branch to "xyz". However, "abc" gets loaded into r12 and the processor branches to "abc".

```
            lda   xyz, r12
            balx  (r12), r12
abc:  addo ....
.
.
.
xyz:  ...
```

IMPLICATION: This erratum reduces the number of registers available for other use.

WORKAROUND: Avoid using the same register for the targ and the dst registers of a **balx** instruction. If the application is composed only of C code compiled by the Intel CTools or GNU C compiler, then this situation will not occur.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600014. Data Breakpoints on System Procedure Entries are Lost for Certain Fault Types

PROBLEM: This erratum occurs when tracing is enabled (PC.te = 1), and a system supervisor fault call is serviced from supervisor mode or a system local fault call is serviced from either user or supervisor mode. If there is a data breakpoint on the fault handler entry in the system procedure table, the breakpoint trace fault is lost.

IMPLICATION: A data breakpoint cannot break upon entering a fault handler pointed to in the system procedure table.

WORKAROUND: Avoid setting data breakpoints on fault handler entries in the system procedure table for the above mentioned fault cases.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

9600015. Actual Max Tov Greater than Specified in Data Sheets

PROBLEM: The actual maximum output valid delay on PQFP packages for all outputs except ALE/ALE# inactive and DT/R# is 18ns at cold and room temperatures (-6 and 25 degrees Celsius). This exceeds the specified value (15ns) in 80960JX data sheets.

IMPLICATION: In systems requiring output valid times less than 18ns, wait states may have to be added.

WORKAROUND: There are no workarounds.

STATUS: Intel plans to fix on subsequent steppings of the 80960JA/JF/JD.

9600016. VIH level on TRST#/RESET# Greater than Specified in Data Sheets

PROBLEM: The input high level voltage (VIH) on PQFP packages for inputs TRST# and RESET# is 2.6V at cold, room and hot temperatures (-6, 25 and 121 degrees Celsius). This exceeds the specified value (2.0V) in 80960JX data sheets.

IMPLICATION: Increased drive is needed to ensure the threshold of the affected signals is exceeded.

WORKAROUND: There are no workarounds.

STATUS: Refer to Summary Table of Changes to determine the affected stepping(s).

SPECIFICATION CHANGES

None for this revision of this specification update.

SPECIFICATION CLARIFICATIONS

None for this revision of this specification update.

DOCUMENTATION CHANGES

001. IPND should not be Modified using a MEM Format Instruction

ISSUE: Table 3-4 on page 3-9 of the user's manual states that the access types for the IPND register include **ld**, **st**, **sysctl**, and **atmod**. It should state that only **atmod** should be used for modification of the IPND register. Using MEM Format instructions to access this register may result in unpredictable behavior.

Intel plans to fix in the next revision of the *i960® Jx Microprocessor User's Manual*.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

002. Page 7-9

ISSUE: The settings specified in the last two sentences on the page were reversed. These sentences should read: "Setting the value to 0 reserves no frames for high-priority interrupts. Setting the value to 7 causes the register cache to become disabled for non-critical code."

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

003. Page 9-10

ISSUE: Fault types not listed in correct order. Also, OPERATION.UNALIGNED is missing from the list. The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

004. Page 9-3, Table 9-1

ISSUE: Incorrect fault record number for 7H PROTECTION fault.

Original text: XX07 XX01H

Corrected text: XX07 XX02H

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

005. Page 12-10, section 12.6.1

ISSUE: Incomplete sentence. Original text: Address bits for are compared...

Corrected text: Only address bits with corresponding MA bits set are compared.

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

006. Page C-23, Figure C-25

ISSUE: LMMR0-1 Register Diagram is a duplicate of LMADR0-1 register. Figure12-5 (page 12-5) displays the register correctly.

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

007. Page 11-29, Example 11-6

ISSUE: Trace Controls section not included in example. The bottom two sections of the part of the example that appeared on page 11-2 is incorrect.

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

008. Page 17-09, Section 17 3.5

ISSUE: Sentence six of the first paragraph of section 17.3.5 incorrectly states:

"The TAP controller is automatically initialized on power-up."

A new paragraph has been added between the first and second paragraphs:

"The TAP controller is not automatically initialized on power-up. Therefore, it is important that the system resets the TAP controller after power up by asserting the TRST# pin. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods. Systems that do not use JTAG, or that normally do not apply a clock to TCK should provide a pull-down resistor on TRST# to hold the TAP controller in the Test_Logic_Reset state. A 2.7k value is strong enough to overcome the TRST# pins internal pull-up, but weak enough to allow automatic test equipment to overdrive it during production testing.

Alternatively, the TRST# pin may be connected to ground if the Test Access Port will never be used."

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

009. Page 13-08, Section 13.2.6.3

ISSUE: Paragraph 2, Sentence 1 formerly read, "The low-order four bits of IMAP0 are used to buffer the expanded-mode interrupt internally."

The sentence now reads: "Do not write to the low-order four bits of IMAP0 as these bits are used to buffer the expanded-mode interrupt internally."

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

010. Page 9-8, Table 9-3

ISSUE: The bottom right cell in Table 9-3 incorrectly reads "Any access." The text in that cell now reads "Break on Data Read or Data Write Access."

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

011. Page 14-10, Section 14.3

ISSUE: The second bulleted item in section 14.3, page 14-10 incorrectly reads:

"- the auto reload is not selected (TMRx.reload=0). See section 14.1.1.1, Bit 0 - Terminal Count Status Bit (TMRx.tc) (pg. 143)"

In section 14.3, the last sentence and its two bulleted items have been deleted. The second sentence of section 14.3 correctly states: "When a timer detects a zero count in its TCR, the timer will force the generation of an internal edge-detected Timer Interrupt signal (TINTx) to the interrupt controller, and the interrupt-pending (IPND.tipx) will be set in the interrupt controller."

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

012. Page 6-57, Section 6.6.32

ISSUE: The action section of the halt instruction (Section 6.6.32 on page 6-57) contains the following incorrect pseudocode:

```
case 0: # Disable interrupts. Clear ICON.gie.
        global_interrupt_enable = false;              break;
case 1: # Enable interrupts. Set ICON.gie.
        global_interrupt_enable = true;               break;
case 2: # Use the current interrupt enable state.     break;
```

The pseudocode now correctly reads:

```
case 0: # Disable interrupts. Clear ICON.gie.
        global_interrupt_enable = true;               break;
case 1: # Enable interrupts. Set ICON.gie.
        global_interrupt_enable = false;              break;
case 2: # Use the current interrupt enable state.     break;
```

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

013. Page 11-13, Example 11-1

ISSUE: In Example 11-1, on page 11-13, the line:

```
for (i=0; i<6; i++) /* carry is carry out from previous add */
```

Has been changed to:

```
for (i=0; i<8; i++) /* carry is carry out from previous add */
```

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

014. Page 3-9, Table 3-4

ISSUE: The allowed access types for the IPND and IMSK registers are listed R/W, AtMod. The corrected text states that the user must use the atmod instruction to modify these registers.

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

015. Page 11-13, Example 11-1

ISSUE: In the original example, the order of the two lines is reversed.:

```
DLMCON.be = (memory[ibr_ptr + 0xc] >> 7);
PMCON14_15[byte2] = 0xc0 & memory[ibr_ptr + 8];
```

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User’s Manual*, Order #272483

016. Page 3-22

ISSUE: The second paragraph originally read: When the processor is reinitialized with a sysctl reinitialize message, the PC register is not changed.

The corrected text reads: When the processor is reinitialized with a sysctl reinitialize message, the PC register returns to its reset value.

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User’s Manual*, Order #272483

017. Pages B-1, B-6 and B7

ISSUE: Tables B-1, B-3 and B-4 show a "T" column for a branch prediction bit. Since branch prediction is not implemented on the i960 Jx processor, this bit is ignored.

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User’s Manual*, Order #272483

018. Page 13-21, Table 13-10

ISSUE: The decision (diamond) incorrectly reads: is ICON.GIE = 1?

The corrected statement reads: is ICON.GIE = 0?

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User’s Manual*, Order #272483

019. Page 2-5, Table 2-4

ISSUE: The bottom right cell originally read:
12345678H (r4)
F0DEBC9AH (r5)

The corrected text reads:
12345678H (r4)
9ABCDEF0H (r5)

The corrected page is appended to this document.

AFFECTED DOCUMENT: *i960® Jx Microprocessor User's Manual*, Order #272483

For example, Table 2-3 shows four bytes of data in memory. Table 2-4 shows the differences between little and big endian accesses for byte, short, word and long word data. Figure 2-2 shows the resultant data placement in registers.

Once data is read into registers, byte order is no longer relevant. The lowest significant bit is always bit 0. The most significant bit is always bit 31 for words, bit 15 for short words, and bit 7 for bytes.

Byte ordering affects the way the i960 Jx processor handles bus accesses. See section 15.2.6, "Byte Ordering and Bus Accesses" (pg. 15-28) for more information.

Errata: 5-13-96 BWL.

Table 2-4 on page 2-5:

The bottom right cell originally read:

12345678H (r4)

F0DEBC9AH (r5)

The corrected text reads:

12345678H (r4)

9ABCDEF0H (r5)

Table 2-3. Memory Contents For Little and Big Endian Example

ADDRESS	DATA
1000H	12H
1001H	34H
1002H	56H
1003H	78H

Table 2-4. Byte Ordering for Little and Big Endian Accesses

Access	Example	Register Contents (Little Endian)	Register Contents (Big Endian)
Byte at 1000H	`ldob 0x1000, r3`	12H	12H
Short at 1002H	`ldos 0x1002, r3`	7856H	5678H
Word at 1000H	`ld 0x1000, r3`	78563412H	12345678H
Long Word at 1000H	`ldl 0x1000, r4`	78563412H (r4) F0DEBC9AH (r5)	12345678H (r4) 9ABCDEF0H (r5)

Table 3-4. Supervisor Space Family Registers and Tables (Sheet 1 of 3)

Register Name	Memory-Mapped Address	Access Type
Reserved	FF00 8000H to FF00 80FFH	—
(DLMCON) Default Logical Memory Configuration Register	FF00 8100H	R/W
Reserved	FF00 8104H	—
(LMADR0) Logical Memory Address Register 0	FF00 8108H	R/W
(LMMR0) Logical Memory Mask Register 0	FF00 810CH	R/W
(LMADR1) Logical Memory Address Register 1	FF00 8110H	R/W
(LMMR1) Logical Memory Mask Register 1	FF00 8114H	R/W
Reserved	FF00 8118H to FF00 83FFH	—
(IPB0) Instruction Address Breakpoint Register 0	FF00 8400H	Sysctl- RwG/WwG
(IPB1) Instruction Address Breakpoint Register 1	FF00 8404H	Sysctl- RwG/WwG
Reserved	FF00 8408H to FF00 841FH	—
(DAB0) Data Address Breakpoint Register 0	FF00 8420H	R/W, WwG
(DAB1) Data Address Breakpoint Register 1	FF00 8424H	R/W, WwG
Reserved	FF00 8428H to FF00 843FH	—
(BPCON) Breakpoint Control Register	FF00 8440H	R/W, WwG
Reserved	FF00 8444H to FF00 84FFH	—
(IPND) Interrupt Pending Register	FF00 8500H	AtMod
(IMSK) Interrupt Mask Register	FF00 8504H	AtMod
Reserved	FF00 8508H to FF00 850FH	—
(ICON) Interrupt Control Word	FF00 8510H	R/W
Reserved	FF00 8514H to FF00 851FH	—
(IMAP0) Interrupt Map Register 0	FF00 8520H	R/W
(IMAP1) Interrupt Map Register 1	FF00 8524H	R/W
(IMAP2) Interrupt Map Register 2	FF00 8528H	R/W
Reserved	FF00 852CH to FF00 85FFH	—

Errata Table 3-4, Page 3-9 (3/8/96) BWL.

The allowed access types for the IPND and IMSK registers include R/W and AtMod.

The corrected text states that the user must use the **atmod** instruction to modify these registers.

When process controls are changed as described above, the processor recognizes the changes immediately except for one situation: if **modpc** is used to change the trace enable bit, the processor may not recognize the change before the next four non-branch instructions are executed.

After initialization (hardware reset), the process controls reflect the following conditions:

- priority = 31
- trace enable = disabled
- execution mode = supervisor
- state = interrupted

When the processor is reinitialized with a **sysctl** reinitialize message, the PC register returns to its reset value.

Normally, **modpc** is not used to modify execution mode or trace fault state flags except under special circumstances, such as in initialization code.

Errata Page 3-22: 3/12/96 BWL:

The second paragraph originally read:

When the processor is reinitialized with a sysctl reinitialize message, the PC register is not changed.

The corrected text reads:

When the processor is reinitialized with a sysctl reinitialize message, the PC register returns to its reset value.

3.6.4 Trace Controls (TC) Register

The TC register, in conjunction with the PC register, controls processor tracing facilities. It contains trace mode enable bits and trace event flags which are used to enable specific tracing modes and record trace events, respectively. Trace controls are described in CHAPTER 10, TRACING AND DEBUGGING.

3.7 USER SUPERVISOR PROTECTION MODEL

The processor can be in either of two execution modes: user or supervisor. The capability of a separate user and supervisor execution mode creates a code and data protection mechanism referred to as the user supervisor protection model. This mechanism allows code, data and stack for a kernel (or system executive) to reside in the same address space as code, data and stack for the application. The mechanism restricts access to all or parts of the kernel by the application code. This protection mechanism prevents application software from inadvertently altering the kernel.

3.7.1 Supervisor Mode Resources

Supervisor mode is a privileged mode which provides several additional capabilities over user mode.

- When the processor switches to supervisor mode, it also switches to the supervisor stack. Switching to the supervisor stack helps maintain a kernel’s integrity. For example, it allows system debugging software or a system monitor to be accessed, even if an application’s program destroys its own stack.

6.2.32 halt (80960Jx-Specific Instruction)

Mnemonic:	**halt**	Halt CPU
Format:	**halt**	*src1* reg/lit

Description: Causes the processor to enter HALT mode which is described in Chapter 16, HALT MODE. Entry into Halt mode allows the interrupt enable state to be conditionally changed based on the value of *src1*.

src1	Operation
0	Disable interrupts and halt
1	Enable interrupts and halt
2	Use current interrupt enable state and halt.

The processor exits Halt mode on a hardware reset or upon receipt of an interrupt that should be delivered based on the current process priority. After executing the interrupt that forced the processor out of Halt mode, execution resumes at the instruction immediately after the **halt** instruction. The processor must be in supervisor mode to use this instruction.

Errata, 4-18-95. BWL.
Section 6.6.32 (pg 6-57)

The action section of the halt instruction contains the following incorrect pseudocode:

case 0: # Disable interrupts. Clear ICON.gie.
global_interrupt_enable = false; break;
case 1: # Enable interrupts. Set ICON.gie.
global_interrupt_enable = true;break;
case 2: # Use the current interrupt enable state.
break;

The pseudocode now correctly reads as shown.

Action:

```
implicit_syncf;
if (PC.em != supervisor)
  generate_fault( TYPE.MISMATCH);
switch(src1) {
    case 0: # Disable interrupts. Clear ICON.gie.
            global_interrupt_enable = true;          break;
    case 1: # Enable interrupts. Set ICON.gie.
            global_interrupt_enable = false;         break;
    case 2: # Use the current interrupt enable state.
            break;
    default:
            generate_fault( OPERATION.INVALID_OPERAND );
            break;
}

ensure_bus_is_quiescient;
enter_HALT_mode;
```

Faults:

STANDARD	Refer to section 6.1.6, "Faults" (pg. 6-6).
TYPE.MISMATCH	Attempt to execute instruction while not in supervisor mode.
OPERATION.INVALID_OPERAND	

The instruction **flushreg**, described in section 6.2.30, “flushreg” (pg. 6-55), is provided to write all local register sets (except the current one) to their associated stack frames in memory. The register cache is then invalidated, meaning that all flushed register sets are restored from their save areas in memory.

For most programs, the existence of the multiple local register sets and their saving/restoring in the stack frames should be transparent. However, some cases where it may not be apparent follow.

- Without executing **flushreg** first, a store to memory does not necessarily update a local register set.
- Without executing **flushreg** first, reading from memory does not necessarily return the current value of a local register set.
- There is no mechanism, including **flushreg**, to access the current local register set with a read or write to memory.
- **flushreg** must be executed sometime before returning from the current frame if the current procedure modifies the PFP in register r0, or else the behavior of the **ret** instruction is not predictable.
- The values of the local registers r2 to r15 in a new frame are undefined.

flushreg is commonly used in debuggers or fault handlers to gain access to all saved local registers. In this way, call history may be traced back through nested procedures.

7.1.4.1 Reserving Local Register Sets for High Priority Interrupts

To decrease interrupt latency for high priority interrupts (interrupted state and process priority greater than or equal to 28), software can limit the number of frames available to all remaining code. This includes code that is either in the executing state (non-interrupted) or code that is in the interrupted state, but, has a process priority less than 28. For the purposes of discussion here, this remaining code will be referred to as *non-critical code*. Specifying a limit for non-critical code, ensures that some number of free frames are available to high-priority interrupt service routines. Software can specify the limit for non-critical code by writing bits 10 through 8 of the register cache configuration word in the PRCB (see Figure 11-6 on page 11-16). The value indicates how many frames within the register cache may be used by non-critical code before a frame needs to be flushed to external memory. The programmed limit is used only when a frame is pushed, which occurs only for an implicit or explicit call.

Allowed values of the programmed limit range from 0 to 7. Setting the value to 0 reserves no frames for high-priority interrupts. Setting the value to 7 causes the register cache to become disabled for non-critical code.

Errata: 11/14/94 BWL

The settings specified in the last two sentences on the page were reversed. These sentences should read:

“Setting the value to 0 reserves no frames for high-priority interrupts. Setting the value to 7 causes the register cache to become disabled for non-critical code.”

The fault handling procedure can optionally use the subtype number to select a specific fault handling action. The i960 Jx processor recognizes i960 architecture-defined faults and a new fault subtype for detecting unaligned memory accesses. Table 9-1 lists all faults that the i960 Jx processor detects, arranged by type and subtype. Text that follows the table gives column definitions.

Table 9-1. i960® Jx Processor Fault Types and Subtypes

Fault Type		Fault Subtype		Fault Record
Number	**Name**	**Number or Bit Position**	**Name**	
0H	OVERRIDE	NA	NA	See section 9.10.1, “Overrides” (pg. 9-21)
0H	PARALLEL	NA	NA	see section 9.6.4, “Parallel Faults” (pg. 9-11)
1H	TRACE	Bit 1	INSTRUCTION	XX01 XX02H
		Bit 2	BRANCH	XX01 XX04H
		Bit 3	CALL	XX01 XX08H
		Bit 4	RETURN	XX01 XX10H
		Bit 5	PRERETURN	XX01 XX20H
		Bit 6	SUPERVISOR	XX01 XX40H
		Bit 7	MARK	XX01 XX80H
2H	OPERATION	1H	INVALID_OPCODE	XX02 XX01H
		2H	UNIMPLEMENTED	XX02 XX02H
		3H	UNALIGNED	XX02 XX03H
		4H	INVALID_OPERAND	XX02 XX04H
3H	ARITHMETIC	1H	INTEGER_OVERFLOW	XX03 XX01H
		2H	ZERO-DIVIDE	XX03 XX02H
4H	Reserved			
5H	CONSTRAINT	1H	RANGE	XX05 XX01H
6H	Reserved			
7H	PROTECTION	Bit 1	LENGTH	XX07 XX02H
8H - 9H	Reserved			
AH	TYPE	1H	MISMATCH	XX0A XX01H
BH - FH	Reserved			

Errata (10-25-94) BWL- Incorrect fault record number for 7H PROTECTION fault.

Original text: XX07 XX01H
Corrected text: XX07 XX02H

In Table 9-1:

- The first (left-most) column contains the fault type numbers in hexadecimal.
- The second column shows the fault type name.

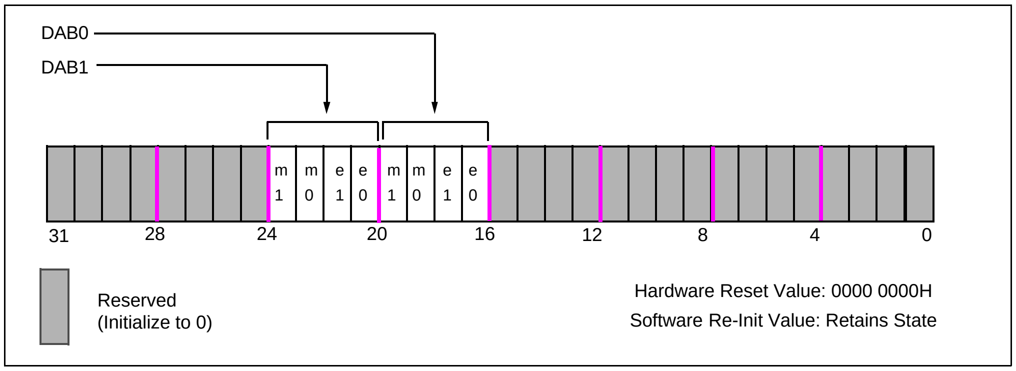

DAB0
DAB1
m 1
m 0
e 1
e 0
m 1
m 0
e 1
e 0
31
28
24
20
16
12
8
4
0
Reserved (Initialize to 0)
Hardware Reset Value: 0000 0000H
Software Re-Init Value: Retains State

Figure 9-2. Breakpoint Control Register (BPCON)

Programming the BPCON register is summarized in Table 9-2.

Table 9-2. Configuring the Data Address Breakpoint Registers

PC.te	DABx.e1	DABx.e0	Description
0	X	X	No action. With PC.te clear, breakpoints are globally disabled.
X	0	0	No action. DABx is disabled.
1	0	1	Reserved.
1	1	0	Reserved.
1	1	1	Generate a Trace Fault.
Note: “X” = don’t care. Reserved combinations must not be used.			

The mode bits of BPCON control what type of access generates a fault, trace message, or break event, as summarized in Table 9-3.

Errata: 2/14/95, BWL.

The bottom right cell in Table 9-3 incorrectly reads “Any access.” The text in that cell now reads “Break on Data Read or Data Write Access.”

Table 9-3. Programming the Data Address Breakpoint Modes

DABx.m1	DABx.m0	Mode
0	0	Break on Data Write Access Only.
0	1	Break on Data Read or Data Write Access.
1	0	Break on Data Read Access.
1	1	Break on Data Read or Data Write Access.

9.6 MULTIPLE AND PARALLEL FAULTS

Multiple fault conditions can occur during a single instruction execution and during multiple instruction execution when the instructions are executed by different units within the processor. The following sections describe how faults are handled under these conditions.

9.6.1 Multiple Non-Trace Faults on the Same Instruction

Multiple fault conditions can occur during a single instruction execution. For example, an instruction can have an invalid operand and unaligned address. When this situation occurs, the processor is required to recognize and generate at least one of the fault conditions. The processor may not detect all fault conditions and may not report all detected faults on a single instruction.

In a multiple fault situation, the reported fault condition is left to the implementation. On the Jx processor, all non-trace fault conditions present in one instruction are prioritized. Only the non-trace fault of highest priority is reported in the fault record. The faults by order of decreasing priority are:

Errata 11/14/94, BWL

Faults types not listed in correct order. Also, OPERATION.UNALIGNED is missing from the list.

- PROTECTION.LENGTH
- OPERATION.UNIMPLEMENTED (Attempt to execute from on-chip RAM or a memory-mapped region only.)
- OPERATION.UNALIGNED
- OPERATION.INVALID_OPCODE
- OPERATION.INVALID_OPERAND
- TYPE.MISMATCH
- OPERATION.UNIMPLEMENTED (All other faults related to unimplemented operations)
- ARITHMETIC.ZERO_DIVIDE
- ARITHMETIC.INTEGER_OVERFLOW
- CONSTRAINT.RANGE

9.6.2 Multiple Trace and Fault Conditions on the Same Instruction

Trace faults on different instructions cannot happen concurrently, because trace faults are precise. Multiple trace fault conditions on the same instruction are reported in a single trace fault record (with the exception of prereturn trace, which always happens alone). To support this multiple fault reporting, the trace fault uses bit positions in the fault-subtype field to indicate occurrences of multiple faults of the same type (Table 9-1).

Example 11-1. Processor Initialization Flow

```
Processor_Initialization_flow()
{  FAIL_pin = true;
   restore_full_cache_mode; disable(I_cache); invalidate(I_cache);
   disable(D_cache); invalidate(D_cache);
   BCON.ctv = 0; /* Selects PMCON14_15 to control all accesses */
   PMCON14_15 = 0; /* Selects 8-bit bus width */

/** Exit Reset State & Start_Init **/
   if (STEST_ON_RISING_EDGE_OF_RESET)
        status = BIST();   /* BIST does not return if it fails */
   FAIL_pin = false;
   PC = 0x001f2002;           /* PC.Priority = 31, PC.em = Supervisor,*/
                              /* PC.te = 0; PC.State = Interrupted    */
   ibr_ptr = 0xfeffff30;      /* ibr_ptr used to fetch IBR words      */

/** Read PMCON14_15 image in IBR **/
   FAIL_pin = true;       IMSK       = 0;
   DLMCON.dcen = 0;       LMMR0.lmte = 0;    LMMR1.lmte = 0;
   PMCON14_15[byte2] = 0xc0 & memory[ibr_ptr + 8];
   DLMCON.be = (memory[ibr_ptr + 0xc] >> 7);

/** Compute CheckSum on Boot Record **/
   carry = 0;  CheckSum = 0xffffffff;
   for (i=0; i<6; i++) /* carry is carry out from previous add*/
        CheckSum = memory[ibr_ptr + 16 + i*4] + CheckSum + carry;
   if (CheckSum != 0)
        { fail_msg = 0xfeffff64;   /* Fail BUS Confidence Test */
          dummy = memory[fail_msg];  /* Do load with address = fail_msg */
          for (;;) ;
          }                          /* loop forever with FAIL pin true */
   else   FAIL_pin = false;

/** Process PRCB and Control Table **/
   prcb_ptr   = memory[ibr_ptr+0x14];
   ctrl_table = memory[prcb_ptr+4];
   Process_PRCB(prcb_ptr);     /* See Process PRCB Section for Details */
   IP = memory[ibr_ptr+0x10];
g0 = DEVICE_ID;
   return;/* Execute First Instruction */
}
```

11

Bit 31 of the assembled PMCON word loaded from the IBR is written to DLMCON.be to establish the initial endianism of memory; the processor initializes the DLMCON.dcen bit to 0 to disable data caching. The remainder of the assembled word is used to initialize PMCON14_15. In conjunction with this step, the processor clears the bus control table valid bit (BCON.ctv), to ensure for the remainder of initialization that every bus request issued takes configuration information from the PMCON14_15 register, regardless of the memory region associated with the request. At a later point in initialization, the processor loads the remainder of the memory region

ERRATA (3-8-96) BWL Page 11-13, Example 11-1

In the original example, the order of the two lines:

```
DLMCON.be = (memory[ibr_ptr + 0xc] >> 7);
PMCON14_15[byte2] = 0xc0 & memory[ibr_ptr + 8];
```

is reversed.

Example 11-1. Processor Initialization Flow

```
Processor_Initialization_flow()
{  FAIL_pin = true;
   restore_full_cache_mode; disable(I_cache); invalidate(I_cache);
   disable(D_cache); invalidate(D_cache);
   BCON.ctv = 0; /* Selects PMCON14_15 to control all accesses */
   PMCON14_15 = 0; /* Selects 8-bit bus width */

/** Exit Reset State & Start_Init **/
   if (STEST_ON_RISING_EDGE_OF_RESET)
       status = BIST();   /* BIST does not return if it fails */
   FAIL_pin = false;
   PC = 0x001f2002;           /* PC.Priority = 31, PC.em = Supervisor,*/
                              /* PC.te = 0; PC.State = Interrupted    */
   ibr_ptr = 0xfeffff30;      /* ibr_ptr used to fetch IBR words      */

/** Read PMCON14_15 image in IBR **/
   FAIL_pin = true;       IMSK       = 0;
   DLMCON.dcen = 0;       LMMR0.lmte = 0;   LMMR1.lmte = 0;
   DLMCON.be = (memory[ibr_ptr + 0xc] >> 7);
   PMCON14_15[byte2] = 0xc0 & memory[ibr_ptr + 8];

/** Compute CheckSum on Boot Record **/
   carry = 0;  CheckSum = 0xffffffff;
   for (i=0; i<8; i++) /* carry is carry out from previous add*/
       CheckSum = memory[ibr_ptr + 16 + i*4] + CheckSum + carry;
   if (CheckSum != 0)
       { fail_msg = 0xfeffff64;   /* Fail BUS Confidence Test */
         dummy = memory[fail_msg];  /* Do load with address = fail_msg */
         for (;;) ;
         }                          /* loop forever with FAIL pin true */
   else   FAIL_pin = false;

/** Process PRCB and Control Table **/
   prcb_ptr   = memory[ibr_ptr+0x14];
   ctrl_table = memory[prcb_ptr+4];
   Process_PRCB(prcb_ptr);     /* See Process PRCB Section for Details */
   IP = memory[ibr_ptr+0x10];
g0 = DEVICE_ID;
   return;/* Execute First Instruction */
}
```

Errata: 12-20-94 BWL.

In Example 11-1, the line:
for (i=0; i<6; i++) /* carry is carry out from previous add*/

Has been changed to:
for (i=0; i<8; i++) /* carry is carry out from previous add*/

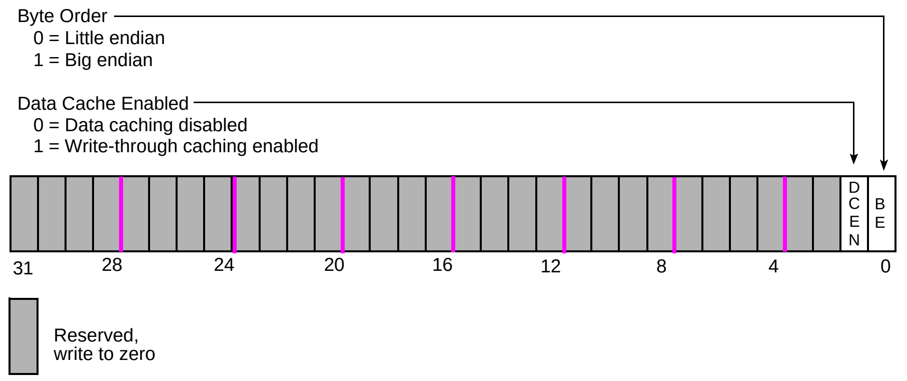
Bit 31 of the assembled PMCON word loaded from the IBR is written to DLMCON.be to establish the initial endianism of memory; the processor initializes the DLMCON.dcen bit to 0 to disable data caching. The remainder of the assembled word is used to initialize PMCON14_15. In conjunction with this step, the processor clears the bus control table valid bit (BCON.ctv), to ensure for the remainder of initialization that every bus request issued takes configuration information from the PMCON14_15 register, regardless of the memory region associated with the request. At a later point in initialization, the processor loads the remainder of the memory region

Example 11-6. Control Table (ctltbl.c)

```
/*----------------------------------------------------------*/
/*  ctltbl.c                                                */
/*----------------------------------------------------------*/
#include "init.h"

typedef struct
   {
   unsigned control_reg[28];
   }CONTROL_TABLE;
const CONTROL_TABLE boot_control_table = {
   /* Reserved */
   0, 0, 0, 0,
   /* Interrupt Map Registers */
   0, 0, 0,/* Interrupt Map Regs (set by code as needed) */
  0x43bc,   /* ICON
             *                - dedicated mode,
             *                - enabled
             * system_init 0 - falling edge actived,
             * system_init 1 - falling edge actived,
             * system_init 2 - falling edge actived,
             * system_init 3 - falling edge actived,
             * system_init 4 - level-low activated,
             * system_init 5 - falling edge actived,
             * system_init 6 - falling edge actived,
             * system_init 7 - falling edge actived,
             *                - mask unchanged,
             *                - not cached,
             *                - fast,
             */

   /* Physical Memory Configuration Registers */

   DEFAULT, 0,   /* Region 0_1 */
   DEFAULT, 0,   /* Region 2_3 */
   DEFAULT, 0,   /* Region 4_5 */
   I_O, 0,       /* Region 6_7 */
   DEFAULT, 0,   /* Region 8_9 */
   DEFAULT, 0,   /* Region 10_11 */
   DRAM, 0,      /* Region 12_13 */
   ROM, 0,       /* Region 14_15 */


   0, 0,         /* Reserved */
   0,            /* Trace Controls */
   1             /* Bus Control Register (Region config. valid) */
};
```

Errata 10-25-94 --BWL.

Added Trace Controls section to Example 11-6.



Mnemonic	Bit/Bit Field Name	Bit Position(s)	Function
DCEN	Data Cache Enable	1	Controls data caching for areas not within other logical memory templates. 0 = Data caching disabled 1 = Write-through caching enabled Instruction caching is never affected by this bit.
BE	Big Endian Byte Order	0	Controls byte order for all accesses, both instruction and data, to memory. 0 = Little endian 1 = Big endian

Figure 12-6. Default Logical Memory Configuration Register (DLMCON)

12.6.1 Defining the Effective Range of a Logical Data Template

For each logical data template, an LMADR register sets the base address using the A31:12 field. The LMMR register sets the address mask using the MA31:12 field. The effective address range for a logical data template is defined using the A31:12 field in an LMADRx register and the MA31:12 field in an LMMRx register. For each access, the upper 20 address bits (A31:12) are compared against A31:12 in the LMADRx register. Only address bits with corresponding MA bits set are compared. Address bits with corresponding MA bits cleared (0) are automatically considered a "match". The processor will only use the logical data template when all compared address bits match. Two examples help clarify the operation of the address comparators.

Errata (10-20-94) BWL- Incomplete sentence.

Original text: Address bits for are compared...

Corrected text: Only address bits with corresponding MA bits set are compared.

13.2.6.3 Mixed Mode

In mixed mode, pins $\overline{\text{XINT0}}$ through $\overline{\text{XINT4}}$ are configured for expanded mode. These pins are encoded for the five most-significant bits of an expanded-mode vector number; the three least-significant bits of the vector number are set internally to 010_2. Pins $\overline{\text{XINT5}}$ through $\overline{\text{XINT7}}$ are configured for dedicated mode.

Do not write to the low-order four bits of IMAP0 as these bits are used to buffer the expanded-mode interrupt internally. $\overline{\text{XINT4:1}}$ are placed in IMAP0[3:0]; $\overline{\text{XINT0}}$ is latched in a special register for use in further arbitrating the interrupt and in selecting the interrupt handler.

Errata 2-3-95, BWL:

Section 13.2.6.3, Paragraph 2, Sentence 1 formerly read, "The low-order four bits of IMAP0 are used to buffer the expanded-mode interrupt internally." The sentence now explicitly tells the user not to write to the low-order four bits of IMAP0.

IMSK register bit 0 is a global mask for the expanded-mode interrupts; bits 5 through 7 mask the dedicated interrupts from pins $\overline{\text{XINT5}}$ through $\overline{\text{XINT7}}$, respectively. IMSK register bits 1-4 must be set to 0 in mixed mode. The IPND register posts interrupts from the dedicated-mode pins $\overline{\text{XINT7:5}}$. IPND register bits that correspond to expanded-mode inputs are not used.

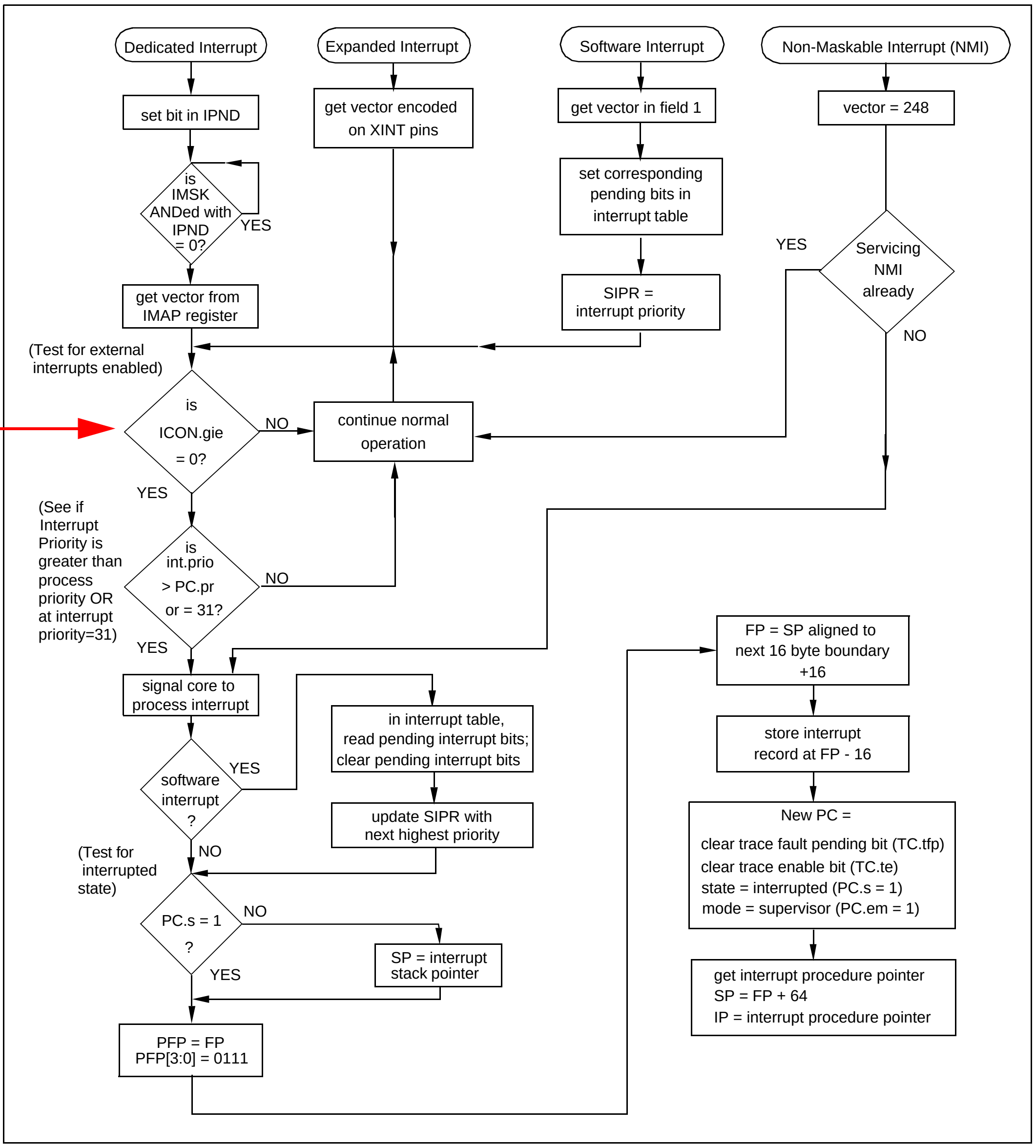
13.2.7 Saving the Interrupt Mask

Whenever an interrupt requested by $\overline{\text{XINT7:0}}$ or by the internal timers is serviced, the IMSK register is automatically saved in register r3 of the new local register set allocated for the interrupt handler. After the mask is saved, the IMSK register is optionally cleared. This allows all interrupts except $\overline{\text{NMI}}$s to be masked while an interrupt is being serviced. Since the IMSK register value is saved, the interrupt procedure can restore the value before returning. The option of clearing the mask is selected by programming the ICON register as described in section 13.3.4, "Interrupt Control Register (ICON)" (pg. 13-12). Several options are provided for interrupt mask handling:

1. Mask is unchanged.
2. Clear for dedicated-mode sources only.
3. Clear for expanded-mode sources only.
4. Clear for all hardware-requested interrupts (dedicated and expanded mode).

Options 2 and 3 are used in mixed mode, where both dedicated-mode and expanded-mode inputs are allowed. Timer unit interrupts are always dedicated-mode interrupts.

Note that if the same interrupt is requested simultaneously by a dedicated- and an expanded-mode source, the interrupt is considered an expanded-mode interrupt and the IMSK register is handled accordingly.



Dedicated Interrupt
Expanded Interrupt
Software Interrupt
Non-Maskable Interrupt (NMI)
set bit in IPND
get vector encoded on XINT pins
get vector in field 1
vector = 248
is IMSK ANDed with IPND = 0?
YES
set corresponding pending bits in interrupt table
Servicing NMI already
YES
NO
get vector from IMAP register
SIPR = interrupt priority
(Test for external interrupts enabled)
is ICON.gie = 0?
NO
continue normal operation
YES
(See if Interrupt Priority is greater than process priority OR at interrupt priority=31)
is int.prio > PC.pr or = 31?
NO
YES
signal core to process interrupt
in interrupt table, read pending interrupt bits; clear pending interrupt bits
software interrupt ?
YES
NO
update SIPR with next highest priority
(Test for interrupted state)
PC.s = 1 ?
NO
YES
SP = interrupt stack pointer
PFP = FP
PFP[3:0] = 0111
FP = SP aligned to next 16 byte boundary +16
store interrupt record at FP - 16
New PC =
clear trace fault pending bit (TC.tfp)
clear trace enable bit (TC.te)
state = interrupted (PC.s = 1)
mode = supervisor (PC.em = 1)
get interrupt procedure pointer
SP = FP + 64
IP = interrupt procedure pointer

Figure 13-10. Interrupt Service Flowchart

13

Errata 4-8-96 BWL

Page 13-21, Table 13-10.

The diamond that reads:

“is ICON.GIE = 1?”

Should read:

“is ICON.GIE = 0?”

Table 17-3. **IEEE Instructions** (Sheet 2 of 2)

Instruction / Requisite	Opcode	Description
bypass IEEE 1149.1 Required	1111_2	**bypass** instruction selects the Bypass register between TDI and TDO pins while in SHIFT_DR state, effectively bypassing the processor's test logic. 0_2 is captured in the CAPTURE_DR state. This is the only instruction that accesses the Bypass register. While this instruction is in effect, all other test data registers have no effect on the operation of the system. Test data registers with both test and system functionality perform their system functions when this instruction is selected.
runbist i960 Jx Processor Optional	0111_2	**runbist** selects the one-bit RUNBIST register, loads a value of 1 into it and connects it to TDO. It also initiates the processor's built-in self test (BIST) feature which is able to detect approximately 82% of the stuck-at faults on the device. The processor AC/DC specifications for V_{CC} and CLKIN must be met and $\overline{\text{RESET}}$ must be de-asserted prior to executing **runbist**. After loading **runbist** instruction code into the instruction register, the TAP controller must be placed in the Run-Test/Idle state. **bist** begins on the first rising edge of TCK after the Run-Test/Idle state is entered. The TAP controller must remain in the Run-Test/Idle state until **bist** is completed. **runbist** requires approximately 414,000 core cycles to complete **bist** and report the result to the RUNBIST register's. The results are stored in bit 0 of the RUNBIST register. After the report completes, the value in the RUNBIST register is shifted out on TDO during the Shift-DR state. A value of 0 being shifted out on TDO indicates **bist** completed successfully. A value of 1 indicates a failure occurred. After **bist** completes, the processor must be recycled through the reset state to begin normal operation.

17.3.5 TAP Controller

The TAP controller is a 16-state synchronous finite state machine that controls the sequence of test logic operations. The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (i.e., PLD) that interfaces to the Test Access Port (TAP). The TAP controller changes state only in response to a rising edge of TCK or power-up. The value of the test mode select (TMS) input signal at a rising edge of TCK controls the sequence of state changes.

The TAP controller is not automatically initialized on power-up. Therefore, it is important that the system resets the TAP controller after power up by asserting the $\overline{\text{TRST}}$ pin. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods. Systems that do not use JTAG, or that normally do not apply a clock to TCK should provide a pull-down resistor on $\overline{\text{TRST}}$ to hold the TAP controller in the Test_Logic_Reset state. A 2.7k value is strong enough to overcome the $\overline{\text{TRST}}$ pin's internal pull-up, but weak enough to allow automatic test equipment to overdrive it during production testing. Alternatively, the $\overline{\text{TRST}}$ pin may be connected to ground if the Test Access Port will never be used.

Errata: BWL (12-16-94)

Sentence 6 of the first paragraph of section 17.3.5 incorrectly stated:

"The TAP controller is automatically initialized on powerup."

A new paragraph (<--left) has been added between the first and second paragraphs.

APPENDIX B
OPCODES AND EXECUTION TIMES

B.1 INSTRUCTION REFERENCE BY OPCODE

This section lists the instruction encoding for each i960 Jx microprocessor instruction. Instructions are grouped by instruction format and listed by opcode within each format.

Table B-1. Miscellaneous Instruction Encoding Bits

M3	M2	M1	S2	S1	Description
					REG Format
x	x	0	x	0	*src1* is a global or local register
x	x	1	x	0	*src1* is a literal
x	x	0	x	1	reserved
x	x	1	x	1	reserved
x	0	x	0	x	*src2* is a global or local register
x	1	x	0	x	*src2* is a literal
x	0	x	1	x	reserved
x	1	x	1	x	reserved
0	x	x	x	x	*src*/*dst* is a global or local register
1	x	x	x	x	*src*/*dst* is a literal when used as a source. M3 may not be 1 when *src*/*dst* is used as a destination only or is used both as a source and destination in an instruction (**atmod, modify, extract, modpc**).
					COBR Format
—	—	0	0	—	*src1 src2* and *dst* are global or local registers
—	—	1	0	—	*src1* is a literal, *src2* and *dst* are global or local registers
—	—	0	1	—	reserved
—	—	1	1	—	reserved

Errata, 4/8/96 BWL:
Tables B-1, B-3 and B-4 show a "T" column for a branch prediction bit. Since branch prediction is not implemented on the i960 Jx processor, this bit is ignored. The "T" column has been removed.

Table B-3. COBR Format Instruction Encodings

Opcode	Mnemonic	Cycles to Execute	Opcode	*src1*	*src2*	M	Displacement	S2
			31........... 24	23 . 19	18... 14	13	12.........2	0
20	**testno**	4	0010 0000	*dst*		M1		S2
21	**testg**	4	0010 0001	*dst*		M1		S2
22	**teste**	4	0010 0010	*dst*		M1		S2
23	**testge**	4	0010 0011	*dst*		M1		S2
24	**testl**	4	0010 0100	*dst*		M1		S2
25	**testne**	4	0010 0101	*dst*		M1		S2
26	**testle**	4	0010 0110	*dst*		M1		S2
27	**testo**	4	0010 0111	*dst*		M1		S2
30	**bbc**	2 + 1[1]	0011 0000	*bitpos*	*src*	M1	*targ*	S2
31	**cmpobg**	2 + 1	0011 0001	*src1*	*src2*	M1	*targ*	S2
32	**cmpobe**	2 + 1	0011 0010	*src1*	*src2*	M1	*targ*	S2
33	**cmpobge**	2 + 1	0011 0011	*src1*	*src2*	M1	*targ*	S2
34	**cmpobl**	2 + 1	0011 0100	*src1*	*src2*	M1	*targ*	S2
35	**cmpobne**	2 + 1	0011 0101	*src1*	*src2*	M1	*targ*	S2
36	**cmpoble**	2 + 1	0011 0110	*src1*	*src2*	M1	*targ*	S2
37	**bbs**	2 + 1	0011 0111	*bitpos*	*src*	M1	*targ*	S2
38	**cmpibno**	2 + 1	0011 1000	*src1*	*src2*	M1	*targ*	S2
39	**cmpibg**	2 + 1	0011 1001	*src1*	*src2*	M1	*targ*	S2
3A	**cmpibe**	2 + 1	0011 1010	*src1*	*src2*	M1	*targ*	S2
3B	**cmpibge**	2 + 1	0011 1011	*src1*	*src2*	M1	*targ*	S2
3C	**cmpibl**	2 + 1	0011 1100	*src1*	*src2*	M1	*targ*	S2
3D	**cmpibne**	2 + 1	0011 1101	*src1*	*src2*	M1	*targ*	S2
3E	**cmpible**	2 + 1	0011 1110	*src1*	*src2*	M1	*targ*	S2
3F	**cmpibo**	2 + 1	0011 1111	*src1*	*src2*	M1	*targ*	S2

1. Indicates that it takes 2 cycles to execute the instruction plus an additional cycle to fetch the target instruction if the branch is taken.

Errata, 4/8/96 BWL:
Tables B-1, B-3 and B-4 show a "T" column for a branch prediction bit. Since branch prediction is not implemented on the i960 Jx processor, this bit is ignored. The “T” column has been removed.

Table B-4. CTRL Format Instruction Encodings

Opcode	Mnemonic	Cycles to Execute	Opcode	Displacement	0
			31............24	23...........2	0
08	**b**	1 + 1[1]	0000 1000	*targ*	0
09	**call**	7	0000 1001	*targ*	0
0A	**ret**	6	0000 1010		0
0B	**bal**	1 + 1	0000 1011	*targ*	0
10	**bno**	1 + 1	0001 0000	*targ*	0
11	**bg**	1 + 1	0001 0001	*targ*	0
12	**be**	1 + 1	0001 0010	*targ*	0
13	**bge**	1 + 1	0001 0011	*targ*	0
14	**bl**	1 + 1	0001 0100	*targ*	0
15	**bne**	1 + 1	0001 0101	*targ*	0
16	**ble**	1 + 1	0001 0110	*targ*	0
17	**bo**	1 + 1	0001 0111	*targ*	0
18	**faultno**	13	0001 1000		0
19	**faultg**	13	0001 1001		0
1A	**faulte**	13	0001 1010		0
1B	**faultge**	13	0001 1011		0
1C	**faultl**	13	0001 1100		0
1D	**faultne**	13	0001 1101		0
1E	**faultle**	13	0001 1110		0
1F	**faulto**	13	0001 1111		0

1. Indicates that it takes 1 cycle to execute the instruction plus an additional cycle to fetch the target instruction if the branch is taken.

Errata, 4/8/96 BWL:
Tables B-1, B-3 and B-4 show a "T" column for a branch prediction bit. Since branch prediction is not implemented on the i960 Jx processor, this bit is ignored. The “T” column has been removed.

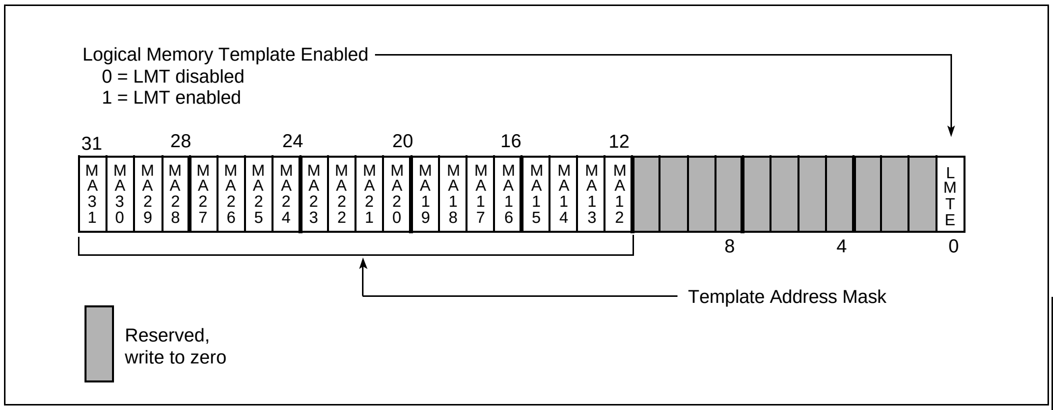


Figure C-25. Logical Memory Template Mask Registers (LMMR0-1)

Errata 11/14/94 BWL.

Figure C-25, LMMR0-1 Register Diagramis a duplicate of LMADR0-1 register. Figure12-5 (page 12-5) displays the register correctly.

Section 12.6, "Programming the Logical Memory Attributes" (pg. 12-8)

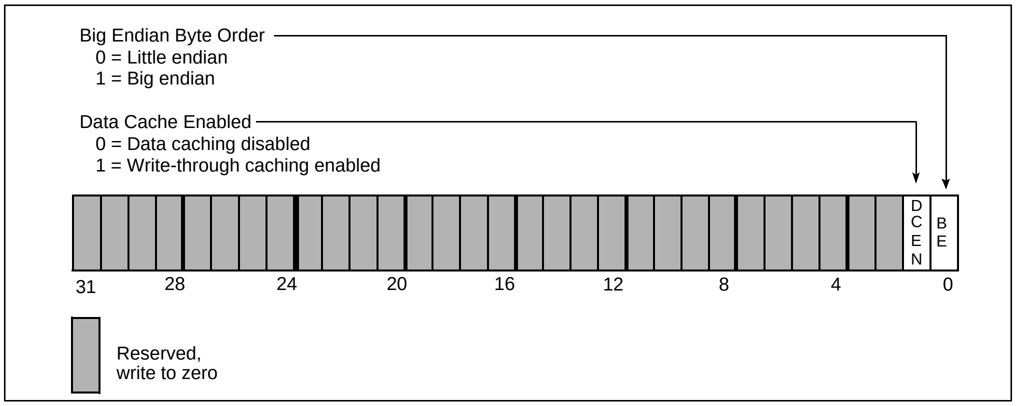


Figure C-26. Default Logical Memory Configuration Register (DLMCON)

Section 12.6, "Programming the Logical Memory Attributes" (pg. 12-8)