80960HA/HD/HT

Specification Update

April 1998

Notice: The 80960HA/HD/HT may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update..

Order Number: 272830-005

Contents

Revision History

Date	Version	Description
04/10/98	005	Added “B2” Stepping to errata table. Added errata item 27. Power Supply Sequence Can Damage Internal Diodes
11/3/97	004	Added errata item 26. Deasserting Level-Detect Interrupts During CISC Instructions Can Produce Spurious INVALID_OPCODE Faults Added Specification Clarification 3. BSTALL Does Not Always Coincide With BREQ. Added Documentation Changes 23 and 24 to incorporate this specification clarification into the 80960Hx documentation.
10/07/97	003	Added errata item 25. DT/R# Timings Do Not Meet Published Specifications. Removed Specification Change items 1–3; these changes have been incorporated into the *80960HA/HD/HT 32-Bit High Performance Superscalar Processor* datasheet.
01/01/97	002	Added descriptions of the B-0 stepping. Added errata items: 20. Using atmod or sysctl to Change IMSK or IPND MMRS Can Hang the Processor 21. Storing the Contents of the I_CACHE to External Memory Also Disables the Cache 22. PCHK# Pin Does Not Indicate Parity Failures On HD and HT Processors 23. Spurious INVALID_OPCODE Faults Can Occur with Level-Detect Interrupts 24. Parity Can Fail on Reliable Data and Can Pass on Corrupted Data Added Document Change items: 5. Page 6-45 13. Page 11-19
07/01/96	001	This is the new 80960Hx Specification Update document. It contains all identified errata published prior to this date.
12/08/95	1.01	Add Specification Clarification 2. Instruction Breakpoints Are Superseded by Invalid Opcode Faults
11/17/95	1.00	The A-1 stepping fixes the following errata from the A-0 (A-0 Errata Sheet numbering shown): 14. RESET Has Priority Over HOLD, 15. Data Cache Global Disable Bit (CCON.dci, sf2) May Take 1 Extra Clock Cycle To Complete, 16. Low Temperature Operating Limit Increased to 25°C, and 17. IPND Register Not Cleared Automatically. Remaining errata have been renumbered for the A-1 stepping. See the Summary of Known Errata, pg. 3 Add errata: #20 Halt Mode Does not Conserve Power, and 13. Invalidating the Data Cache Automatically Re-enables It Modifications since rev 0.11 (10/27/95) of the A-0 errata sheet... #8 Hold V_{CC} Above 3.15 V and Below 3.45 V #15 DEN# Remains Asserted During ADS# Cycles #16 TRST# Input Can Be Tied Low, and #17 Burst Accesses on 8- and 16-Bit Buses Do Not Behave Like the Cx Processor

Preface

As of July, 1996, Intel's Computing Enhancement 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 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 information that was not previously published.

Affected Documents/Related Documents

Title	Order
80960HA/HD/HT 32-Bit High Performance Superscalar Processor datasheet	272495
i960® Hx Microprocessor User's Manual	272484
i960® Hx Microprocessor Instruction Set and Register Quick Reference	272792
AP-506: Designing for 80960Cx and 80960Hx Compatibility	272556
Reduced Power Options for the 80960HA/HD/HT Processor[1]	

1. Can be downloaded from the Intel worldwide web homepage at: http://www.intel.com/design/i960/technote/hxlopwr.htm

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 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, or documentation changes which apply to the Product Name 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

Stepping

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
	A0	A1	A2	B0	B1	B2			
1	X	X	X				8	Fixed	Parity Failure on 8- and 16-bit Unaligned Loads
2	X	X	X	X	X	X	8	NoFix	Read Wrong Location from Non-Burst, 8- and 16-bit Memory Regions
3	X	X	X	X	X	X	9	NoFix	Breakpoints on Stacks Produce Wrong Fault IP
4	X	X	X				9	Fixed	Parity Faults May Not Report Correct Address and Access Type
5	X	X	X				9	Fixed	PMCON15 Temporarily Initialized Incorrectly During RESET
6	X	X	X				10	Fixed	BCON Register is not Cleared Before Software Reset
7	X	X	X				10	Fixed	MODTC Command Can Set TC Register Event Flags
8	X	X	X	X	X	X	10	NoFix	Parity Faults Cannot Be Disabled Separate from the PCHK# Pin
9	X	X	X				10	Fixed	Timer Terminal Count (TMR.tc) Bit Cannot Bear Polling
10	X	X	X	X	X	X	11	NoFix	Return Instruction Pointer (RIP) Cannot be Stored by Software
11	X	X	X				11	Fixed	WAIT# Pin Asserts During NXDA Wait States
12	X	X	X				12	Fixed	Software Interrupts Can Access the Wrong Handler Address
13	X	X	X				12	Fixed	Invalidating the Data Cache Automatically Re-enables It
14	X						13	Fixed	RESET Has Priority Over HOLD
15	X						13	Fixed	Data Cache Global Disable Bit (CCON.dci, sf2) May Take 1 Extra Clock Cycle To Complete
16	X						14	Fixed	Low Temperature Operating Limit Increased to 25°C
17	X						14	Fixed	IPND Register Not Cleared Automatically
18	X	X	X				15	Fixed	Cycle Type Bits (CT3:0) Do Not Indicate Some Fault Types
19	X	X	X	X	X	X	16	NoFix	Operation Fault Occurs When Clearing the IMASK (sf1) Register
20	X	X	X				17	Fixed	Using atmod or sysctl to Change IMSK or IPND MMRS Can Hang the Processor
21	X	X	X				18	Fixed	Storing the Contents of the I_CACHE to External Memory Also Disables the Cache
22	X	X	X				18	Fixed	PCHK# Pin Does Not Indicate Parity Failures On HD and HT Processors
23	X	X	X	X	X	X	18	NoFix	Spurious INVALID_OPCODE Faults Can Occur with Level-Detect Interrupts
24	X	X	X	X			21	Fixed	Parity Can Fail on Reliable Data and Can Pass on Corrupted Data
25	X	X	X	X	X	X	22	NoFix	DT/R# Timings Do Not Meet Published Specifications
26	X	X	X	X	X	X	24	NoFix	Deasserting Level-Detect Interrupts During CISC Instructions Can Produce Spurious INVALID_OPCODE Faults
27	X	X	X	X	X	X	27	No Fix	Power Supply Sequence Can Damage Internal Diodes

Specification Changes

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

Specification Clarifications

No.	Steppings			Page	Status	SPECIFICATION CLARIFICATIONS
	#	#	#			
1				29		Burst Accesses on 8- and 16-Bit Buses Do Not Behave Like the Cx Processor
2				30		Instruction Breakpoints Are Superseded by Invalid Opcode Faults
3				30		BSTALL Does Not Always Coincide With BREQ

Documentation Changes

No.	Document Revision	Page	Status	DOCUMENTATION CHANGES
1	272484	32		Page 3-11, Table 3-4
2	272484	32		Page 3-11, Table 3-4
3	272484	32		Page 3-26
4	272484	32		Page 4-6, Section 4.4.3
5	272484	32		Page 6-45
6	272484	32		Page 6-60, Table 6-8
7	272484	33		Page 6-61, Figure 6-4
8	272484	33		Page 6-62, Table 6-9
9	272484	33		Page 6-63, Figure 6-5
10	272484	33		Page 6-64, Figure 6-6
11	272484	33		Page 6-116
12	272484	33		Page 8-6, Figure 6-6
13	272484	34		Page 11-19
14	272484	34		Page 12-15
15	272484	34		Page 13-4, Figure 13-2
16	272484	34		Page 13-9, Section 13.2.2.5
17	272484	34		Page 13-11, Section 13.2.2.5
18	272484	35		Page 13-23, Table 13-7
19	272484	35		Page 13-37, Figure 13-9
20	272484	35		Page 13-37, Figure 13-10
21	272484	35		Page 15-15
22	272484	35		Pages E-41, E-44, E-45, Examples E-1, E-3, E-4
23	272556	35		AP-506, Page 17, “BSTALL” Section, 4th Paragraph
24	272495	36		Datasheet, Page 8, Table 6

Identification Information

Stepping Register

80960HA/HD/HT processors may be identified electrically according to device type and stepping. The *g0* register contains this information after reset initialization. The following table lists the devices to which this errata sheet applies:

Table 1. Device Identifier Codes Found in the g0 Register After Reset

	Device		
Stepping	**80960HA**	**80960HD**	**80960HT**
A-0	0x08840013	0x08841013	0x08842013
A-1 and A-2	0x18840013	0x18841013	0x18842013
B-0, B-1, and B-2	0x28840013	0x28841013	0x28842013

Refer to the data sheet for instructions on how to obtain the identifier number from the *g0* register.

JTAG Registers

See the datasheet, release -006, dated July 1997, Section 5.1 80960Hx Boundary Scan Chain, pages 74-77 for the boundary scan chain definition.

See the data sheet, release -006, dated July 1997, Section 5.2 Boundary Scan Description Language Example (BSDL) for the simulator file describing the boundary scan configuration in the PGA and PQ2 packages. Contact your Intel sales office for an ASCII version of these files. Optionally, these BSDL files can be downloaded from the Intel worldwide web homepage at: http://www.intel.com/design/i960/swsup/ .

See the user’s manual, release -001, dated November 1995, Section 16.2 Boundary Scan (JTAG) for a full description of the implemented boundary scan registers and instructions.

Errata

1. Parity Failure on 8- and 16-bit Unaligned Loads

Problem: The parity detection logic can falsely indicate a parity failure under specific conditions.

Implication: False parity failures may result during unaligned short reads on an 8- or 16-bit bus.

Parity must be enabled on that 8- or 16-bit memory region for the failure to occur. Also, the error does not appear if the 8- or 16-bit memory region is designated as cacheable.

This error does not affect 32-bit memory regions.

Workaround: One or more of the following conditions will prevent this error:

- Disable parity for 8- or 16-bit memory regions containing unaligned data.
- Make the 8- or 16-bit memory region cacheable. (Since cacheable loads are always promoted to word loads, the errata conditions never occur.)
- Do not use short loads (***ldis*** or ***ldos***) on unaligned data in 8- or 16-bit regions. If necessary, break short loads into two discrete byte loads.

These workarounds do not necessarily have to be removed after this errata is corrected in silicon.

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

2. Read Wrong Location from Non-Burst, 8- and 16-bit Memory Regions

Problem: Under certain conditions, the processor reads a wrong memory location when reading unaligned data from either an 8- or 16-bit memory region.

The failure mode occurs when all the following conditions are present:

- Bursting disabled
- Pipelining enabled
- 8- or 16-bit memory region
- $N_{RAD} = 0$ and $N_{RDD} \neq 0$
- Unaligned memory read access that crosses a 16-byte (quad word) boundary

If any of the above conditions are not present, the processor behaves correctly.

When the above conditions are present, the processor may fail to access the correct location in the next 16-byte memory segment. Instead, it may “wrap around” and access a wrong location at the beginning of the current 16-byte segment.

Implication: There is little impact to the user since it is impractical to design a pipelined memory systems using NRAD = 0 with NRDD ≠ 0.

Workaround: In every 8- or 16-bit memory region where bursting is disabled and pipelining is enabled, set NRAD ≠ 0 or NRDD = 0. Else, avoid at least one of the other conditions listed above.

Status: **NoFix.** Refer to Summary Table of Changes to determine the affected stepping(s).

3. Breakpoints on Stacks Produce Wrong Fault IP

Problem: When a data breakpoint is set on a stack location and a ***call***, ***callx***, or ***calls*** instruction causes a flush to that stack location, the resulting trace fault record may report the instruction pointer (IP) of the called procedure instead of the calling instruction.

This error occurs only when the procedure call causes a frame flush from the on-chip register cache to the procedure stack.

Implication: The IP returned for breakpoints set on stack locations is unreliable.

Workaround: Avoid setting data breakpoints on the stack. Else, ensure that the register cache is large enough to prevent frame spills during debugging.

Otherwise, ignore the fault IP if you only need to know that data was flushed to the stack.

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

4. Parity Faults May Not Report Correct Address and Access Type

Problem: When a parity fault occurs, the fault record may report the wrong faulting address and bus access type. Specifically, if another load or fetch access immediately follows the faulting access, the fault record address and bus access type describes the second access instead of the faulting access.

Implication: The faulting address and bus access type in a parity fault record are not reliable.

Workaround: Ignore the address of faulting instruction and the access type word of the parity fault record.

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

5. PMCON15 Temporarily Initialized Incorrectly During RESET

Problem: The PMCON15 bytes loaded from the Initialization Boot Record (IBR) after RESET is deasserted become corrupted inside the processor. The resulting wait state profile can cause initialization read accesses to have more address-to-data (NRAD) and data-to-data (NRDD) wait states than intended.

This problem corrects itself later during initialization when the processor overwrites PMCON15 with the correct wait state profile from the Control Table image in user memory.

Specifically, the low nibble of IBR PMCON Byte 1 is logically OR'd with the high nibble of PMCON Byte 0.

The write wait states in Byte 1 are not at issue here because no writes occur during processor initialization after PMCON15 is overwritten from the Control Table.

Implication: If the workaround is ignored, some systems may "hang" indefinitely during processor initialization. Memory systems that use READY# during processor initialization cannot afford arbitrary extra wait states because the processor ignores the READY# signal until after the wait states expire. In that case, the processor can "hang" during initialization, awaiting a READY# signal that has already occurred.

Workaround: Program IBR address 0xFEFFFF34 with 0x00.

No workaround is strictly necessary for memory systems that use the internal wait state generator. Processor initialization proceeds correctly, but possibly at a slower speed until the processor loads the Control Table from external memory.

This workaround does not necessarily have to be removed once this errata is corrected in silicon.

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

6. BCON Register is not Cleared Before Software Reset

Problem: Processor microcode does not clear the BCON.sirp bit before performing a *sysctl* software reset.

Implication: A TYPE MISMATCH fault is generated if the BCON.sirp bit is set when a software reset is executed.

Workaround: Clear BCON.sirp before executing a ***sysctl*** reset sequence.

This workaround does not necessarily have to be removed after this errata is corrected in silicon.

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

7. MODTC Command Can Set TC Register Event Flags

Problem: The *modtc* instruction can be used to set event flags in the Trace Control (TC) Register. Normally, event flags are set by hardware trace events and cleared by user software with *modtc*. There is no utility in the user setting those flags.

Implication: User code could accidentally set the TC Register event flags with unpredictable results.

Workaround: Only use ***modtc*** to clear event flags.

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

8. Parity Faults Cannot Be Disabled Separate from the PCHK# Pin

Problem: Contrary to section 16.3.5 “Parity Generation and Checking” in the user’s manual revision -001 (dated November 1995), parity faults cannot be disabled independently from the hardware parity checking pin, PCHK#. There is no bit in the PRCB Fault Configuration Word to enable/disable faults on parity errors.

Implication: When parity is enabled, parity faults and the PCHK# pin responds to parity failures. Users cannot independently disable one or the other response.

Workaround: Under evaluation.

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

9. Timer Terminal Count (TMR.tc) Bit Cannot Bear Polling

Problem: The TMRx.tc bit randomly fails to go true (high) if polled by software when the timer is used in one-shot mode. Timer0 and Timer1 are both affected.

Specifically, if the user software reads TMRx.tc at or about the same time the bit is set by the processor, the bit never gets set. The timer expires and halts as normal.

Implication: This errata affects applications that use the timer(s) to produce finite, one-shot delays. Applications that require cyclic, periodic delays can usually use the timer interrupts instead of polling.

Workaround: Use either of the following techniques:

1. Poll the Timer Count Register (TCRx) until it decrements to zero. In one-shot mode, TCRx remains cleared when it reaches 0x0000000.
2. Poll the Timer Enable bit (TMRx.en) until it clears. In one-shot mode, TMRx.en clears when TCRx reaches 0x00000000.

These workarounds do not necessarily have to be removed after this errata is corrected in silicon.

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

10. Return Instruction Pointer (RIP) Cannot be Stored by Software

Problem: A fault occurs when writing the RIP (located in register r2) directly to an external address using the following code sequence.

```
lda <address>, r6  # the register used is not significant

st  RIP, (r6)
```

The following code sequence does not produce a fault.

```
mov RIP, r7         # mov and lda execute in parallel

                    # (1 clock cycle)


lda <address>, r6

st  r7, (r6)
```

Implication: Storing the RIP to external memory is a common debug method, but rarely used in actual applications. Of course, user software should never modify (write) the RIP directly. Section 7.2 “Modifying the PFP Register” in the user’s manual, revision -001 (dated November 1995) describes the recommended way to change the processor’s context.

Workaround: Use an intermediate register to write the RIP from *r2* to an external address.

This workaround does not necessarily have to be removed after this errata is corrected in silicon.

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

11. WAIT# Pin Asserts During NXDA Wait States

Problem: The WAIT# pin toggles true (low) during internally generated NXDA wait states. These extra WAIT# signals occur only when a bus request requires multiple bus accesses.

Implication: Applications that use WAIT# to derive a write data strobe can generate sporadic strobes between valid memory accesses.

Workaround: If your application uses WAIT# to qualify write strobes, modify your write strobe logic to ignore any WAIT# signals after BLAST# and before ADS#. A 1-bit state machine is sufficient. Add the equivalent of the following ABEL logic equations to your strobe logic:

```
write_en       := ads # (write_en & !blast);

write0_out     =  write0 & write_en;

write1_out     =  write1 & write_en;

write2_out     =  write2 & write_en;

write3_out     =  write3 & write_en;
```

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

12. Software Interrupts Can Access the Wrong Handler Address

Problem: Posting a *sysctl* software interrupt to a vector ending in 0xa while vector caching is enabled causes the processor to begin executing at an undefined address, which usually results in an OPERATION fault. The processor fetches an interrupt handler address from the internal vector cache, where it should not.

This behavior occurs every time vector caching is enabled and the vector least significant nibble is 0xa, i.e., the set of “bad” vectors is:

0x0a, 0x1a, 0x2a, 0x3a, 0x4a, 0x5a, 0x6a, 0x7a,

0x8a, 0x9a, 0xaa, 0xba, 0xca, 0xda, 0xea, 0xfa

This failure does not occur when either of the above conditions is false -- when vector caching is disabled or another vector besides 0xa is used.

Interrupt vectors ending in 0xa are not cacheable, so the processor should read the external interrupt vector table even though vector caching is enabled. When the failure occurs, the processor doesn’t read the handler address from the external interrupt vector table.

Expanded or mixed hardware interrupts can use these vectors with impunity. For example, vector 0xaa has been shown to work correctly as an expanded hardware interrupt vector.

Implication: Sixteen (16) software interrupt vectors (all vectors ending in 0xa) are unavailable while vector caching is enabled. The remaining 224 software interrupt vectors are unaffected.

Workaround: Disable interrupt vector caching (ICON.vce = 0) when posting software interrupts to vectors ending in 0xa. Otherwise, avoid using vectors ending in 0xa.

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

13. Invalidating the Data Cache Automatically Re-enables It

Problem: Invalidating the data cache (“D_cache”) enables the D_cache.

Applications that disable the D_cache then invalidate it result in the D_cache being enabled again. This behavior occurs regardless of whether the software directly writes to the CCON (sf2) or uses the ***dcctl*** instruction to manipulate the D_cache.

Implication: The D_cache can be enabled when users do not expect it.

Workaround: Follow one of the sequences below to invalidate and disable the D_cache:

1. Set CCON.dci = 1 to invalidate the D_cache.
2. Loop on the CCON.dci bit until it clears.
3. Set CCON.dcgd = 1 to disable the D_cache.

or

1. Issue the ***dcctl*** instruction, mode 2 to invalidate the D_cache first.
2. Issue the ***dcctl*** instruction, mode 0 to disable the D_cache.

This workaround does not necessarily have to be removed after this errata is corrected in silicon.

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

14. RESET Has Priority Over HOLD

Problem: If the RESET and HOLD pins are both true, the processor output and I/O pins assume the RESET state. The output and I/O pins are supposed to remain in the HOLD state regardless of the RESET pin.

Implication: Single bus master systems are not affected.

Multiple bus master systems that require the 80960Hx processor to remain in HOLD mode during RESET must use the workaround.

Workaround: Prevent the RESET and HOLD pins from being active at the same time.

If a multiple bus master system uses HOLD independent of the system RESET signal, add external logic to qualify the system RESET signal with the HOLD signal.

RESET#sys + HOLD = RESET#proc

where:

RESET#sys (active low) -- system RESET signal from the host or other controlling bus master,

HOLD (active high) -- HOLD signal applied to the processor,

RESET#proc (active low) -- processor RESET pin.

If the latter option is used, ensure that RESET#proc remains asserted for at least 16 bus clock cycles after HOLD goes away to provide enough time to properly reset the processor.

This workaround does not necessarily have to be removed after this errata is corrected in silicon.

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

15. Data Cache Global Disable Bit (CCON.dci, sf2) May Take 1 Extra Clock Cycle To Complete

Problem: Sometimes the CCON.dci bit stays high one extra clock cycle. The processor randomly appears to take an extra clock cycle to invalidate the data cache. Functionally, the bit still works as specified; it may simply take longer.

A hardware race condition in the processor causes the bit to discharge in 2 clock cycles. Setting this bit is not affected.

Implication: Probably no impact. In normal use, user software sets CCON.dci and polls it until it is cleared by the processor, signaling that the invalidation has completed (see Section 4.5.1 in the user's manual, revision -001, dated November 1995). Since most software does not measure the elapse time of those operations, you may not see this condition.

Workaround: Do not count the number of cycles required to invalidate the data cache. Otherwise, no workaround is required.

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

16. Low Temperature Operating Limit Increased to 25°C

Problem: At low temperatures (about -5°C), the on-chip PLL clock circuitry has been observed to lose lock and oscillate unpredictably on a portion of units tested. When this failure condition occurs, the processor behavior becomes unpredictable.

Implication: Unpredictable processor behavior.

Workaround: Avoid operating the processor below 25°C case temperatures.

Status: Fixed. This limitation was screened during production testing at the factory and fixed on all steppings after the A-0 step. (Refer to Summary Table of Changes to determine the affected stepping(s).)

17. IPND Register Not Cleared Automatically

Problem: Sometimes the processor does not automatically clear the Interrupt Pending (IPND) register when servicing a dedicated interrupt. The interrupt itself is still handled correctly. When the failure condition occurs, the interrupt service routine (ISR) keeps executing repeatedly without further interrupt requests until the Interrupt Mask (IMSK) or IPND are cleared.

This failure condition does not appear on every device, and is more pronounced at high V_{CC} voltages. This behavior has been observed at V_{CC} as low as 3.33 V.

Implication: Unless corrected, ISRs can be invoked indefinitely by one dedicated interrupt event.

Workaround: Manually clear the IPND register during the dedicated interrupt ISR. Intel recommends all applications that use dedicated interrupts implement this workaround since the failure condition may appear on some but not all devices.

Status: Fixed. This failure condition has been fixed on the A-1 and all subsequent steps. Refer to Summary Table of Changes to determine the affected stepping(s).

18. Cycle Type Bits (CT3:0) Do Not Indicate Some Fault Types

Problem: Bit CT2 does not go high for certain types of faults.

The table below summarizes the fault conditions when the CT2 bit does and does not work. All cases of each fault subtype are implied to either work correctly or not unless otherwise noted.

The table below summarizes fault conditions when the CT2 bit does and does not work.

AULT TYPE	CT2 BIT WORKS FOR...	
0H Parallel	PARALLEL	
1H Trace	INSTRUCTION	PRERETURN
	BRANCH	
	CALL	
	RETURN	
	SUPERVISOR	
	MARK/BREAKPOINT	MARK/BREAKPOINT
	breakpoint always works correctly, *mark* when Mark Trace Mode is not set in the TC register.	*mark* when Mark Trace Mode is set in the TC register, *fmark.*
2H Operation	INVALID_OPCODE	
	UNIMPLEMENTED	UNIMPLEMENTED
		sysctl message type 04H
	UNALIGNED	
	INVALID_OPERAND	INVALID_OPERAND
	non-existent *sfr*, unaligned long-, triple-, or quad-register, undefined register, writing to RIP.	undefined *sysctl*, *icctl*, *dcctl*, or *intctl* operand.
3H Arithmetic	INTEGER_OVERFLOW	INTEGER_OVERFLOW
		integer divide overflow (*divi*)
	ZERO_DIVIDE	
5H Constraint	RANGE	RANGE
	all other cases.	only if a *fault<cc>* test evaluates true.
7H Protection	BAD_ACCESS	BAD_ACCESS
	GMU detection	GMU protection
		LENGTH
8H Machine	PARITY_ERROR	
AH Type	MISMATCH	MISMATCH
	execute a privileged instruction while in user mode (*intdis, inten*),	execute a privileged instruction while in User mode (*modpc*, *sysctl*, *icctl*, *dcctl, intctl*).
	write to Supervisor MMR while in User mode,	
	access an sfr while in User mode,	
	write to internal RAM with BCON.irp set and in User mode,	
	User write to timer register when timer is protected against User writes,	
	write to the first 64 bytes of internal RAM with BCON.sirp set (User and Supervisor modes).	
10H Override	OVERRIDE	

CT2 does behave correctly on interrupts.

Implication: The CT3:0 bits do not reliably indicate fault code execution. This condition should not affect most applications since the CT3:0 bits are typically only used during development and diagnosis on most applications and by emulator systems.

Workaround: None available. Do not rely on the CT3:0 bits to indicate faults.

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

19. Operation Fault Occurs When Clearing the IMASK (sf1) Register

Problem: An INVALID_OPCODE operation fault occurs on the microcoded instruction when an interrupt occurs within 6-7 bus clock cycles before any of the following sequences:

1. Clear_IMASK_sfr_register microcoded instruction
2. Clear_IPND_sfr_register microcoded instruction
3. Clear_IMSK_sfr_Bit_for_Posted_Interrupt microcoded instruction
4. Clear_IPND_sfr_Bit_for_Posted_Interrupt microcoded instruction

In cases “C” and “D”, clearing bits for inactive interrupts does not cause the failure mode. Setting IMSK or IPND bits does not exhibit the failure mode, either.

A “microcoded instruction” is any assembly language instruction that executes a CISC microcode sequence. Examples include ***call, ret, sysctl, dcctl, atmod, atadd***, most branches, and ***flushreg***. The key to preventing this failure mode is to insert at least 3 RISC instructions, such as *nops* (*mov g0, g0*), after clearing all or part of the IMSK or IPND special function registers.

When the anomalous fault occurs, an interrupt request input occurs within 6-7 bus clock cycles before the bit-clearing instruction sequence. The interrupt can be either external or internal.

The instruction cache, data cache, interrupt vector, interrupt service routine (ISR) caching, Supervisor/User mode, process priority and interrupted/executing state are insignificant to this failure mode.

When the failure condition occurs, the fault handler does execute properly. Then the ISR also executes properly after the fault handler. All subsequent interrupts execute correctly, too.

Other special function registers that also appear as memory mapped registers (MMRs) -- *sf2* (CCON), *sf3* (ICON), and *sf4* (GCON) -- are not affected by this failure condition.

Implication: The fault adds an unexpected time delay in the program execution, and depending on the INVALID_OPCODE fault handler, can unnecessarily redirect the program execution by attempting to recover from an invalid error.

Workaround: Use the three new instructions (***intctl***, ***intdis***, and ***inten***) to globally enable and disable the interrupts before manipulating the IMSK or IPND register. These instructions ensure that the new processor state is in full effect before the instruction completes.

```
intdis
Clear_IMASK_or_IPND_sfr_bits
inten
```

This sequence takes 13 core clock cycles -- no more, no less -- and occupies 3 words of execution code. It guarantees the processor will not service any masked interrupts after the ***intdis*** instruction is issued.

An alternative is to insert at least three *nop* (*mov g0, g0*) instructions as shown below.

```
Clear_IMASK_or_IPND_sfr_bits
nop
nop
nop
resume normal instruction sequence
```

This sequence uses no less than 4 core clock cycles and occupies 4 words of execution code. The maximum execution time is indeterminate because the processor may service an interrupt request masked in the first instruction during any point in the sequence until the normal instruction sequence resumes.

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

20. Using atmod or sysctl to Change IMSK or IPND MMRS Can Hang the Processor

Problem: When an interrupt signal occurs in the vicinity of an *atmod* or *sysctl* instruction acting on the IMSK or IPND memory mapped registers (MMR), the processor can hang.

"Hang" means no further ADS# strobes occur, A31:2 and BE#3:0 maintain their last valid values, and D31:0 float. The only way to recover is through hardware reset or cycle V_{CC} off-on according to the data sheet specifications.

This errata affects the HA, HD, and HT processors.

If an interrupt arrives up to about 15 clocks before the ***atmod*** or ***sysctl*** executes, the failure can occur. The failure does not occur every time in the test environment, though. A minimum of 1 interrupt is required to produce the failure.

When the failure occurs, the interrupt request is never serviced. It's unknown whether the interrupt request is posted correctly in the IPND register when the failure occurs.

Using the bit manipulation instructions (***setbit***, ***clrbit***, etc.) on the special function register manifestations of IPND (*sf0*) and IMSK (*sf1*) does not produce the failure. As defined, ***sysctl*** and ***atmod*** are not compatible with special function registers.

Other ***sysctl*** operations don't produce the failure.

Any interrupt pulse of 3 bus clock cycles long or longer can cause this problem.

This failure has been observed only while the data cache and instruction cache are disabled. Interrupts are enabled, vector cache disabled, and inputs are debounced.

When using dedicated mode, manipulating the IMSK or IPND MMRs can produce the failure. However, when using expanded mode, only the IMSK MMR can produce the problem since expanded interrupts do not affect IPND.

Implication: Using the atmod or sysctl instructions to modify the MMR implementations of IMSK or IPND can hang the processor indefinitely.

Workaround: Manipulate the IPND and IMSK registers using the special function registers (*sf0* = IPND and *sf1* = IMSK). See errata #19 "Operation Fault Occurs When Clearing the IMSK (*sf1*) Register" for related workarounds.

User applications are not well served by ***atmod*** or ***sysctl*** on MMRs that are also special function registers because those operations take several clock cycles to complete. Changes to IMSK and IPND as sfrs complete in 1 clock cycle.

C compilers typically do not generate code to modify IMSK or IPND directly; those instructions must typically be coded as in-line assembly by the user.

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

21. Storing the Contents of the I_CACHE to External Memory Also Disables the Cache

Problem: Case #6 of the *icctl* instruction flushes the Instruction Cache (I_cache) contents to external memory. After flushing, though, the *icctl* instruction disables the I_cache.

No other cases of ***icctl*** exhibit this problem. The ***dcctl*** instruction does not either.

Implication: Since ***icctl*** case #6 is predominantly used for I_cache analysis and system debugging, no impact on production systems is expected.

During system development, users could see abnormally slow system performance after storing the I_cache contents because the I_cache is disabled.

Workaround: Re-enable the I_cache after flushing the I_cache contents with the ***icctl*** case #6 instruction. While there are several ways to implement this workaround, the following instruction sequence will do the trick quickly.

```
icctl 6, src2, src/dst  # flush the I_cache contents to src2
setbit 30, sf2, sf2     # re-enable the I_cache through CCON
resume normal instruction sequence
```

The ***setbit*** instruction executes in 1 clock cycle.

This workaround does not necessarily have to be removed after this errata is fixed in silicon.

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

22. PCHK# Pin Does Not Indicate Parity Failures On HD and HT Processors

Problem: When enabled, a parity failure produces a PARITY_ERROR fault, but does not always assert the PCHK# pin. This problem affects the HD and HT processors.

Implication: The PCHK# pin does not work reliably on the HD and HT processors.

Workaround: None available. Do not rely on the PCHK# pin on the HD or HT processors.

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

23. Spurious INVALID_OPCODE Faults Can Occur with Level-Detect Interrupts

Problem: Spurious INVALID_OPCODE faults can occur on systems using level-detect hardware interrupts. The fault record points to a user instruction that in fact is a valid opcode. The spurious fault occurs when the level-detect interrupt signal on the XINT pins deasserts within a few clock cycles of an interrupt service routine (ISR) *ret* instruction.

To review the terminology, the processor recognizes level-detect interrupts as long as the XINT pins are held low. This mode contrasts to the *falling edge detect* mode which recognizes interrupts only when they transition from high-to-low. Level-detect mode requires the interrupt source to remain asserted until the user explicitly dismisses it in the ISR software.

The hardware interrupt contributing to this problem can be either dedicated level-detect or expanded mode. (All expanded mode interrupts are level-detect by definition.)

This problem affects the HA, HD and HT processors. Enabling or disabling the I_cache, D_cache or register cache may modulate the problem. The clock speed is not a direct factor.

This failure resembles errata #19.“Operation Fault Occurs When Clearing the IMASK (*sf1*) Register”. Both problems appear when the processor recognizes an interrupt request that subsequently disappears during the same microcoded instruction. The processor enters a metastable state and tries to execute a value from an internal lookup table as an instruction. The value is not a valid opcode, so, an INVALID OPCODE fault results. The fault record mistakenly points to a valid user instruction opcode as the cause of the fault.

Implication: Unless prevented by the workarounds, level-detect interrupts can produce spurious INVALID OPCODE faults on valid user instructions.

Workaround: In general terms, prevent interrupts from being recognized and disappearing during the same microcoded (CISC) instruction.

In practical terms, make sure the level-detect interrupt request is gone before the ISR ***ret*** instruction. Any or all of the following workarounds can accomplish this objective –

1. dead reckoning - calculate the worst case latencies.
2. delay the ISR until the interrupt is dismissed - wait until the dismissing bus access completes before proceeding with the ISR execution.
3. poll until the interrupt is gone - poll an interrupt flag until it indicates the interrupt has retired.

DEAD RECKONING -

This option allows you to guarantee the workaround conditions by deductive reasoning instead of by direct control. Therefore, the dead reckoning option requires that the bus access delays are entirely predictable so a worst case timing condition can be calculated. The 80960Hx processor must be the only bus master in the system (the HOLD and BOFF# signals are not used) and the bus wait states must be deterministic (the READY# signal is not controlled by unpredictable outside events). If bus access delays are not entirely predictable, use another workaround.

In essence, you ensure the interrupt is gone before the ***ret*** instruction by calculating and comparing the time each process takes. In mathematical terms:

$T_{RET} > T_{DIS}$

where T_{RET} = the minimum time elapse from issuing the dismiss interrupt instruction to the ***ret*** instruction, and

T_{DIS} = the maximum time elapse between issuing the dismiss interrupt instruction and when the interrupt actually goes away.

The "dismiss interrupt" instruction can be a load or store to an external logic device that causes it to withdraw the interrupt signal from the XINT pins.

Calculate T_{RET} by summing the external bus clock cycles for the shortest path to the ***ret***. Assume the I_cache and D_cache are enabled if the application uses them. Take the internal clock multiplier (HA=1x, HD=2x, HT=3x) into account.

Measuring T_{RET} with a logic analyzer is a little tricky because it is difficult to see when the instructions are actually issued. The user's manual explains that issued bus instructions may not execute right away depending on the condition of the on-chip bus controller.

Calculate T_{DIS} by summing the external bus clock cycles for the longest possible delay between issuing the dismiss interrupt instruction and the interrupt actually retiring. Include as many of the following in the calculation as applicable:

- maximum memory wait state profiles for the regions being accessed
- predictable memory access delays (such as DRAM refresh cycles)
- possible delays from bus requests already pending in the bus queue
- response latency of the external interrupt device

Of course, dismiss the interrupt as early in the ISR as practical.

DELAY THE ISR UNTIL THE INTERRUPT IS DISMISSED -

This workaround eliminates ambiguous delays caused by external bus masters.

Some applications involve multiple bus masters such as DMA controllers that can deny the 80960Hx processor access to the bus for indefinite periods of time. These delays can arbitrarily extend the time the interrupt request remains on the XINT pins while the ISR executes at normal speed from the I_cache. In some cases the interrupt request exceeds the ***ret*** instruction and causes a spurious INVALID OPCODE fault.

This workaround delays ISR execution until the 80960Hx processor regains control of the bus and dismisses the interrupt source. The ***syncf*** instruction delays execution indefinitely until the bus and instruction fetch queues empty. Then execution proceeds again as normal. Add the following code as early in your ISR as practical.

```
#Dismiss external interrupt source dismiss_interrupt_instruction

#Wait until the dismiss_interrupt_instruction executes syncf

#Resume ISR execution
```

You still have the responsibility to ensure the interrupt signal will have enough time to retire before the ***ret*** instruction. (See “DEAD RECKONING”, above.)

POLL UNTIL THE INTERRUPT IS GONE -

This workaround relies on an interrupt request flag bit that can be polled by the user. In operation, you dismiss the external interrupt source as early in the ISR as practical, proceed with the ISR execution, then, just before the ***ret*** instruction, poll on that flag bit until the interrupt request retires.

The 80960Hx processor provides a built-in flag bit for dedicated mode interrupts, but the user’s system has to provide one for expanded mode interrupts.

DEDICATED MODE: The user’s manual, section 11.7.2 “Interrupt Detection Options” offers a polling method (Example 11-5) for dedicated mode level-sensitive interrupts which delays the ***ret*** instruction until the dedicated interrupt request deasserts. The polling code example (with some minor clean-up modifications) appears below. The example assumes that the ***ld*** from address “INTR_SRC” deactivates the XINT7 interrupt input. The loop tries to clear the IPND bit for XINT7 but the bit remains set until the XINT7 interrupt retires.

```
# Clear level-detect interrupts before return from handler
    ld  INTR_SRC, g0   # Dismiss the extern. interrupt
    lda IPND_MMR, g1   # g1 = IPND MMR address
    lda 0x80, g2       # g2 = mask to clear XINT7 IPND bit

# Loop until IPND bit 7 clears
wait:
    mov 0,g3
    # Try to clear the XINT7 IPND bit
    atmodg1, g2, g3
    bbs 0x7, g3, wait  # Branch until IPND bit 7 clears

# Optionally restore IMSK
    mov r3, IMSK

    ret                # Return from handler
```

EXPANDED MODE: Expanded mode interrupts do not post bits in IPND, so an internal polling loop isn't available. The user's system must provide a flag bit on the external interrupt controller. One suggestion is a read/write register on the interrupt controller that indicates the state of the signals being applied to the 80960Hx XINT pins. Poll on that external register until it indicates the interrupt has been retired.

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

24. Parity Can Fail on Reliable Data and Can Pass on Corrupted Data

Problem: Under certain conditions, reliable data (data with correct parity) can fail for parity. Conversely, corrupted data (data with wrong parity) can pass for parity. A parity failure asserts the PCHK# pin and produces a PARITY_ERROR fault.

Memory read accesses that involve multiple data bus widths (32-, 16-, and/or 8-bit) can exhibit this problem.

When transitioning from wider to narrower data bus width (e.g. 32- to 16-bit, or 16- to 8-bit) memory reads, the processor can test the parity of the undefined bits on the narrower data bus. For example, a 32-bit read followed by a 16-bit read can cause the processor to mistakenly test the parity of the undefined bits (bits D31:16) of the data bus. Another example is a 16-bit read followed by an 8-bit read. In that case, the processor can mistakenly test the undefined bits D15:8. Since those undefined bits may randomly include wrong parity, the processor can produce an invalid parity failure.

When transitioning from narrower to wider (e.g., 16- to 32-bit, or 8- to 16-bit) memory reads, the processor sometimes disregards the parity of the upper half of the data bus bits. Any wrong parity in those upper bits goes unnoticed so the processor indicates no failure even when one is warranted.

Implication: The parity check logic cannot be trusted to indicate reliable and corrupted data parity under all operating conditions.

Workaround: Avoid mixing data bus width reads. Otherwise, do not rely on the parity check feature.

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

25. DT/R# Timings Do Not Meet Published Specifications

Problem: The DT/R# T_{OV2} maximum and T_{TVEL} minimum timings exceed the published specs.

This errata affects only the clock-multiplied HD and HT versions; the HA is unaffected. The timings are based on the clock multiple, as shown in Tables 1 and 2.

Table 1. 80960HD DT/R# Delays

		Specified		Actual		
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Units**
T_{OV2}, T_{OH2}	Output Valid Delay and Output Hold for DT/R#	T/2 + 1.5	T/2 + 9.5	T**3/4** + 1.5	T**3/4** + 9.5	ns
T_{TVEL}	DT/R# Valid to DEN# Falling	T/2 – 4		T**/4** – 4		ns

Table 2. 80960HT DT/R# Delays

		Specified		Actual		
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Units**
T_{OV2}, T_{OH2}	Output Valid Delay and Output Hold for DT/R#	T/2 + 1.5	T/2 + 9.5	T**5/6** + 1.5	T**5/6** + 9.5	ns
T_{TVEL}	DT/R# Valid to DEN# Falling	T/2 – 4		T**/6** – 4		ns

Implication: Data bus contentions can occur if the application meets ALL of the following criteria:

1. using HD or HT processor, AND
2. using external bi-directional data bus transceivers, AND
3. controlling the direction of the data bus transceivers with DT/R#, AND
4. the data bus transceivers are too slow to switch direction within the actual T_{TVEL} time shown in Tables 1 and 2.

Condition 4. can be expressed as follows:

$T_{IS}(DIR) > T_{TVEL}$

where:

$T_{IS}(DIR)$ is the transceiver minimum DIRECTION input setup requirement with respect to the transceiver OUTPUT ENABLE input, and

T_{TVEL} is the actual 80960Hx processor minimum “DT/R# Valid to DEN# Falling” output setup time.

Transceiver manufacturers such as Integrated Device Technology (IDT) and Texas Instruments (TI) rarely specify $T_{IS}(DIR)$, so the evaluation may require your engineering judgment or more information from the transceiver manufacturer.

The contentions can happen when the tardy DT/R# signal causes the bi-directional data transceivers to briefly drive in the wrong direction before switching to the correct direction. While driving the wrong direction, the transceivers can contend with either the processor or the memory and I/O data bus.

The contentions can occur only on the first data transferred during a write/read transition (a read-following-a-write or a write-following-a-read) access. Thus, only the first access in a burst is affected.

In general, data bus contentions can produce data corruption and system crashes, and may compromise device reliability. The functionality and reliability issues are explained below.

1. Functional failures (data corruption and system crashes)

   Only zero wait state (NRAD=0 and NWAD=0) memory systems that use data bus transceivers can experience data corruption and system crash failures due to this errata.

   The output valid time (T_{OV1}) of the processor and of the memory system can extend by the length of the contention. For example, an application that uses the 80960HT-75 processor and IDT 74FCT245 transceivers could experience about 1 ns of contention. In this case, the HT T_{OV1} time could extend by 1 ns. Functional failures can occur if that extension violates memory setup times.

   As always, bus contentions can increase noise in the contending devices. The current transients can also introduce V_{CC} noise that can affect the noise margin in other parts of the application. The magnitudes and effects of these perturbations must be understood for each application.

2. Reliability risks

   The worst-case conditions evaluated by Intel using the i960® HT-75 processor and the IDT 74FCT245 transceivers indicate the i960 Hx processor reliability is not compromised by this errata.

   In general, sufficiently large bus contentions can compromise the reliability of the contending components. Excessive average currents in the output drivers can wear them out prematurely. The effect on reliability is a function of average current and the output driver circuit design, layout, and silicon fabrication process.

   The i960 Hx component is designed for at least 100x higher average currents than our studies indicate this errata produces. Using the above example, an application that combines the i960 HT-75 processor and IDT 74FCT245 transceivers could experience a 1 ns contention of 20 - 30 mA every 4 µs. The average current is 5 - 10 µA, well within the i960 Hx processor design limits.

   The reliability of the transceivers and memory and I/O chips may be similarly unaffected. Contact the manufacturers of those chips for further information.

   Depending on your application reliability testing, you may already have quantified the reliability impact in your application.

Workaround: Analyze the timings associated with your application to determine whether the DT/R# timings introduce bus contentions. If no contentions occur, no further workaround is necessary.

If you find contentions, your application can function properly if at least one NRAD and NWAD wait state is inserted to allow the contentions to subside and the bus to stabilize before latching the first data of a read or write sequence. Additional V_{CC} decoupling may be required to suppress power supply transients caused by the contentions according to common engineering practice. Satisfy yourself that the reliability is not compromised.

Alternately, you can synthesize a surrogate DT/R# signal in off-chip logic that closely approximates the original timing specs. The logic produces a DT/R# signal that mimics the W/R# output delayed by T/2. The pseudo code appears below.

```
clk    = !Hx_clkin          ;synch to falling edge of CLKIN

new_DTR =: Hx_WR            ;new_DTR mimics W/R#, delayed by T/2
```

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

26. Deasserting Level-Detect Interrupts During CISC Instructions Can Produce Spurious INVALID_OPCODE Faults

Note: This errata is identical to errata #23, but covers the general case conditions that can produce the failure.

Problem: Spurious INVALID_OPCODE faults can occur on systems that use level-detect hardware interrupts. The fault record points to a user instruction that in fact is a valid opcode. The spurious fault occurs when a pending level-detect interrupt signal on the XINT pins deasserts within a few clock cycles of a microcoded (CISC) instruction that enables that interrupt.

In the case of errata #23, the ***ret*** instruction is the enabling CISC instruction. While concluding an interrupt service routine (ISR), the ***ret*** instruction enables the interrupt that prompted the ISR. If that interrupt remains pending, then deasserts during the ***ret*** execution, the failure occurs.

Any microcoded CISC instruction that can enable pending level-sensitive interrupts is subject to this failure. Examples include the following:

- ***modpc*** (when used to lower the process priority, and thereby enable pending interrupts)
- inten

To review the terminology, the processor recognizes *level-detect* interrupts as long as the XINT pins are held low. *Falling edge-detect* mode recognizes interrupts only as they transition from high to low. Level-detect mode requires the interrupt source to remain asserted until the user explicitly dismisses it in the ISR software.

The hardware interrupt contributing to this failure can be either dedicated level-detect or expanded mode. (All expanded mode interrupts are level-detect by definition.) This problem affects the HA, HD, and HT processors. Enabling or disabling the I_cache, D_cache or register cache may modulate the problem. The clock speed is not a direct factor.

Implication: Unless prevented by the workarounds, level-detect interrupts can produce spurious INVALID_OPCODE faults on valid user instructions.

Workaround: In general terms, prevent pending level-sensitive interrupts from disappearing during the microcoded instructions that enable the interrupts.

In practical terms, make sure the level-detect interrupt signal either remains firmly asserted throughout, or else is completely withdrawn at least 5 CLKIN cycles before, an enabling CISC instruction. Asserting the interrupt signal throughout the CISC instruction invokes an ISR execution.

Either or both of the following workarounds can accomplish this objective:

- dead reckoning – calculate the worst case latencies
- runtime pause – deterministically delay the enabling CISC instruction until the pending interrupt withdraws

DEAD RECKONING

This option prevents the failing conditions by applying deductive reasoning instead of by asserting direct runtime control. The bus access delays must be entirely predictable. The 80960Hx processor must be the only bus master in the system (the HOLD and BOFF# signals are not used), and the bus wait states must be deterministic (the READY# signal is not affected by unpredictable delays). If bus access delays are not entirely predictable, use another workaround.

Ensure the interrupt is gone at least 5 CLKIN cycles before issuing the enabling CISC instruction by comparing the time each process takes. In mathematical terms,

TCISC - (5 CLKIN cycles) > TDIS

where:

TCISC = the minimum time elapse from issuing the dismiss interrupt instruction to the enabling CISC instruction, and

TDIS = the maximum time elapse from issuing the dismiss interrupt instruction to the interrupt signal actually going away.

The “dismiss interrupt” instruction can be any instruction to external logic (typically a load or store) that causes the interrupt signal to withdraw from the XINT pins.

Calculate TCISC by summing the external bus clock cycles for the shortest path to the enabling CISC instruction. Assume the I_cache and D_cache are enabled if the application uses them. Take the internal clock multiplier (HA=1x, HD=2x, HT=3x) into account.

Measuring TCISC with a logic analyzer is tricky because it is difficult to see when the instructions are actually issued. The user’s manual explains that issued bus instructions may not execute right away depending on the condition of the on-chip bus controller.

Calculate TDIS by summing the external bus clock cycles for the longest possible delay between issuing the dismiss interrupt instruction and the interrupt actually retiring. Include as many of the following in the calculation as applicable:

- maximum memory wait state profiles for the regions being accessed
- predictable memory access delays (such as DRAM refresh cycles)
- possible delays from bus requests already pending in the bus queue
- response latency of the external interrupt device

RUNTIME PAUSE

This workaround accounts for indefinite bus access delays by forestalling the enabling CISC instruction until the interrupt signal withdraws.

Some applications involve multiple bus masters, such as DMA controllers, that can deny the 80960Hx processor access to the bus for indefinite periods of time. These delays can arbitrarily extend the time the interrupt request remains on the XINT pins while the ISR executes at normal speed from the I_cache. In some cases, the interrupt request can overlap the enabling CISC instruction, causing the spurious INVALID_OPCODE fault.

This workaround delays execution until the 80960Hx processor regains control of the bus and dismisses the interrupt source. Three methods can suspend execution pending a bus access: syncf, scoreboarding, and polling. syncf postpones execution indefinitely until the bus and instruction fetch queues run empty. Scoreboarding accomplishes the same result by forcing the processor to wait for external data. Polling waits until a data value indicates that the interrupt has withdrawn.

The following pseudo-code demonstrates the syncf option.

```
#Dismiss external interrupt source
dismiss_interrupt_instruction
#Wait until the dismiss_interrupt_instruction executes
syncf
#Resume execution
```

The scoreboarding option operates almost the same way.

```
#Dismiss external interrupt source
ld         <int_cntlr_addr>,r7
#Wait until the bus access occurs
cmpobne    0x0,r7,<next_addr>
#Resume execution
```

You still must ensure that the interrupt signal has enough time to retire before the enabling CISC instruction issues. (See “Dead Reckoning,” above.)

Polling relies on an interrupt request flag bit that can be polled by the processor.   Dismiss the external interrupt source, proceed with normal execution, then, just before the enabling CISC instruction, poll on that flag bit until the interrupt request retires.

The 80960Hx processor provides a built-in flag bit for dedicated mode interrupts, but the user’s system must provide one for expanded mode interrupts.

DEDICATED MODE: The user’s manual, section 11.7.2 “Interrupt Detection Options” offers a polling method (Example 11-5) for dedicated mode level-sensitive interrupts which delays a ***ret*** instruction until the dedicated interrupt request deasserts. The polling code example (with some minor clean-up) appears below. The example assumes that the ld from address “INTR_SRC” deactivates the XINT7 interrupt input. The loop tries to clear the IPND bit for XINT7 but the bit remains set until the XINT7 interrupt retires.

```
# Clear level-detect interrupts before return from handler
    ld  INTR_SRC, g0   # Dismiss the extern. interrupt
    lda IPND_MMR, g1   # g1 = IPND MMR address
    lda 0x80, g2       # g2 = mask to clear XINT7 IPND bit
# Loop until IPND bit 7 clears
wait:
    mov 0,g3
    # Try to clear the XINT7 IPND bit
    atmodg1, g2, g3
    bbs 0x7, g3, wait
    # Optionally restore IMSK
    mov r3, IMSK
    ret                # Return from handler
```

For illustration, this example uses the ***ret*** instruction as the enabling CISC instruction; any other enabling instruction can replace ***ret***.

EXPANDED MODE: Expanded mode interrupts do not post bits in IPND, so an internal polling loop isn’t available. The user’s system must provide a flag bit on the external interrupt controller. One suggestion is a readable register on the interrupt controller that indicates the state of the signals being applied to the 80960Hx XINT pins. Poll on that external register until it indicates that the interrupt has been retired.

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

27. Power Supply Sequence Can Damage Internal Diodes

Problem: If the voltage on the VCCPLL power supply pin exceeds the V_{CC} pin voltage by 0.5 V at any time, including the power up and power down sequences, excessive currents can permanently damage on-chip electrostatic discharge (ESD) protection diodes. The damage can accumulate over multiple episodes.

Pragmatically, this problem only occurs when the VCCPLL and V_{CC} pins are driven by separate power supplies or voltage regulators. Applications that use one power supply for VCCPLL and V_{CC} are not typically at risk. Verify that your application does not allow the VCCPLL voltage to exceed V_{CC} by 0.5 V.

ESD diodes connect the VCCPLL circuitry to V_{CC}. Normally, those diodes are unbiased or reverse biased, so no current flows. In the event of a positive electrostatic pulse on VCCPLL, the diodes protect the phase-locked loop circuitry by shunting the excess charge to V_{CC}. However, when power supplies forward bias these diodes for any length of time the current flow can damage or destroy the diodes.

The VCCPLL low-pass filter recommended in the data sheet does not promote this problem.

The VCC5 power supply pin is also susceptible to excessive current damage, but is adequately protected by the 100 ohm series resistor recommended in the data sheet. See the 80960HA/HD/HT data sheet for more details.

Implication: Diode damage can manifest itself as...

- resistive short circuits between the VCCPLL and V_{CC} pins,
- compromised ESD protection on the VCCPLL pin, and
- unspecified functional or parametric failures resulting from damage to the circuitry near the diodes.

Workaround: Use one common power supply or regulator for the VCCPLL and V_{CC} pins. Otherwise, ensure that the V_{CC} pins power up before VCCPLL and power down after VCCPLL. Else limit the VCCPLL diode current flow by providing at least 100 ohm of resistance in series with the VCCPLL pin.

Status: No Fix. Refer to Summary Table of Changes to determine the affected stepping(s).

Specification Changes

None for this revision of this specification update.

Specification Clarifications

1. Burst Accesses on 8- and 16-Bit Buses Do Not Behave Like the Cx Processor

Issue: The burst behavior on 8-bit and 16-bit buses is more sophisticated on the 80960HA/HD/HT (“Hx”) processors than the 80960CA/CF (“Cx”) processors.

The basic definition of a burst access, is 2-4 consecutive data cycles following a single address cycle. A 1 data cycle burst is impossible.

8-bit bus differences: Whereas the Cx can only burst beginning at a byte-aligned boundary (A1:0 = 0x0), the 80960Hx can begin a burst at any of three places (A1:0 = 0x0, 0x1 or 0x2). Of course, a byte access beginning at A1:0 = 0x3 is a single byte access, not a burst. The table below illustrates this point.

8-Bit Bus Behavior for a Word Access

	Word 0					Word 1					80960Hx Accesses
Access beginning on...	0	1	2	3		4	5	6	7		
... Byte 0 (aligned)	▨	▨	▨	▨							Burst 4 bytes
... Byte 1		▨	▨	▨		▨					Burst 3 bytes, 1 byte
... Byte 2			▨	▨		▨	▨				Burst 2 bytes, burst 2 more bytes
... Byte 3				▨		▨	▨	▨			1 byte, burst 3 bytes

16-bit bus differences: Whereas the Cx maintains the same data type (byte or short) throughout a burst, the 80960Hx can dynamically change data types within a burst. Specifically, a multiple short word burst beginning on an odd byte boundary (A2:1 = 0x1 or 0x3) will produce a burst of a byte followed by a short, or visa versa. The table below illustrates this point.

16-Bit Bus Behavior for a Word Access

	Word 0					Word 1					80960Hx Accesses
Access beginning on...	0	1	2	3		4	5	6	7		
... Byte 0 (aligned)	▨	▨	▨	▨							Burst 2 shorts
... Byte 1		▨	▨	▨		▨					Burst byte & short, byte
... Byte 2			▨	▨		▨	▨				Short, short (no burst)
... Byte 3				▨		▨	▨	▨			Byte, burst short & byte

Implication: This behavior difference is a problem only when off-chip memory control logic assumes even boundary bursting or consistent data types within a burst.

Memory systems that assume even boundary bursting generate the 2 least significant address bits themselves. One such system has been observed to “wrap around” the address, and overwrite unintended memory addresses.

Memory systems that assume consistent data types within a burst fail to recognize subsequent data type changes and fail to access all intended bytes.

Workaround: Do not generate the 2 least significant address bits in your external 8-bit bus memory controller. Rather, pass the processor address bits through to the memory for proper sequencing. Otherwise, disable bursting in memory controllers from incrementing the 2 least significant bits.

Also, pass BE3#, BE1#, and BE0# to your external 16-bit bus memory systems instead of latching these signals on the first burst access.

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

FROM/TO REFERENCE: Section 15.3.2 Burst Accesses, page 15-14. Insert the description and impact text above (sans the "Description" and "Impact" titles) to the bottom of the page, before Figure 15-5 on the following page.

2. Instruction Breakpoints Are Superseded by Invalid Opcode Faults

Issue: An instruction breakpoint on an address containing an invalid opcode does not produce a trace fault; the breakpoint never "breaks". Instead, the opcode produces an INVALID_OPCODE fault.

Implication: This behavior appears only when breakpoints are used, usually limited to system development and diagnostic sessions.

Workaround: Do not set instruction breakpoints on addresses that contain invalid opcodes. More practically, do not set instruction breakpoints on uninitialized or unimplemented memory. This workaround applies to software debug tools as well as user-generated code.

For user-generated code, the GMU (Guarded Memory Unit) offers a better method to protect an uninitialized or unimplemented memory region from accidental accesses.

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

FROM/TO REFERENCE: Section 9.5.2.4 Tracing on Return from Implicit Call: Fault Case, page 9-15. Insert the following text after the first paragraph.

"There is a special case of this behavior. If an instruction breakpoint is set on an address containing an invalid opcode, the processor services the INVALID_OPCODE fault and never services the trace fault."

3. BSTALL Does Not Always Coincide With BREQ

Issue: Neither the datasheet nor the user's manual describe the functional behavior of the BSTALL (Bus Stall) and the BREQ (Bus Request) output signals. Some product literature implies that BSTALL coincides with BREQ, but such is not always the case.

The processor can stall (BSTALL asserted) even with an empty bus queue (BREQ deasserted). Depending on the instruction stream and memory wait states, the two signals can be separated by several CLKIN cycles.

Implication: Bus arbitration logic that logically "ANDs" BSTALL and BREQ will not correctly grant the bus to the processor in all stall cases, potentially degrading processor performance.

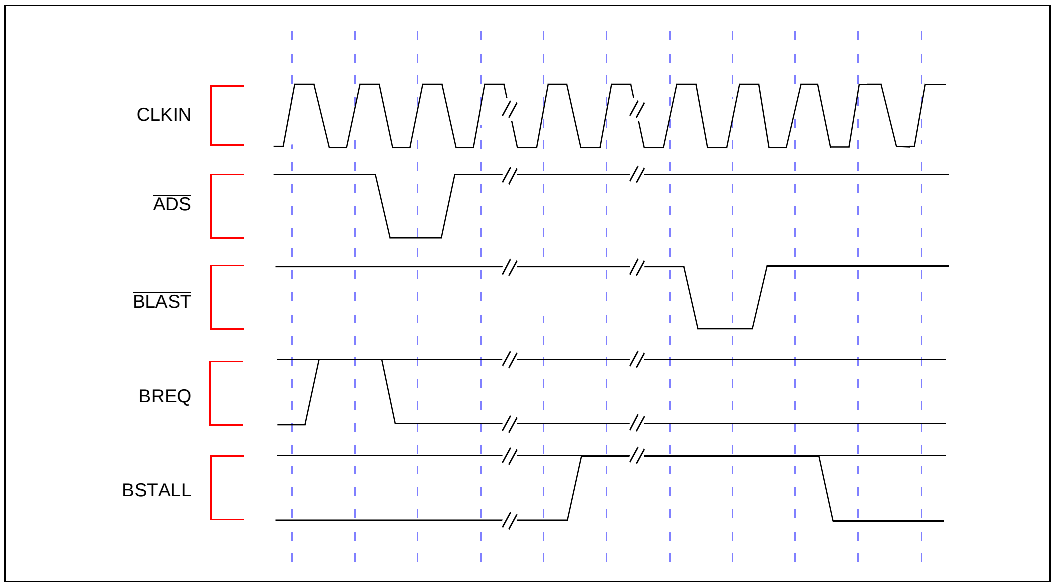
Workaround: Do not logically "AND" BSTALL and BREQ together in arbitration logic. Instead, the simplest bus arbitration should logically "OR" BSTALL and BREQ to determine the processor's bus ownership requirements.

More sophisticated arbitration should recognize the priority nature of these two signals. Using a traffic light analogy, BREQ is a "yellow light" warning of a possible processor stall and BSTALL is a "red light" indicating a stall in progress.

Affected Docs: The *80960HA/HD/HT 32-Bit High-Performance Superscalar Processor* datasheet, August 1997, release 006 and all earlier releases.

FROM/TO REFERENCE: Add Figure 59, as shown:

Figure 59. BREQ and BSTALL Operation



Documentation Changes

1. Page 3-11, Table 3-4

Issue: The original text states that 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.

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

2. Page 3-11, Table 3-4

Issue: The Access Type listed for the BPCON and XBPCON registers originally read:

"R/W, WwG"

The corrected entries read:

"WwG"

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

3. Page 3-26

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.

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

4. Page 4-6, Section 4.4.3

Issue: Sentence added to the end of the first paragraph reads:

"Any code can be locked, not just interrupt routines."

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

5. Page 6-45

Issue: Case 8 was inadvertently omitted from previous revision of the reference document.

The added text reads:

```
case 8:         # invalidate the lines that came from LMTs that had DCIIR set
                # at the time the line was allocated.
                # NOTE : for compatibility with future products that have
                # several independent regions, the value of src2 should be one.
                   invalidate_DCIIR_lines_in_DCache;
                   break;
```

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

6. Page 6-60, Table 6-8

Issue: The word "blocks" is replaced with "ways".

Affected Docs: The *i960® Hx Microprocessor User's Manual*, November 1995, release 001.

7. Page 6-61, Figure 6-4

Issue: Under “Src/Dst Format for I_cache Locking Status”, constant fixed values for bits 0 - 23 have been added.

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

8. Page 6-62, Table 6-9

Issue: The original value for number of ways was listed as 256. The corrected value is 128.

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

9. Page 6-63, Figure 6-5

Issue: Each way should have 8 words, as opposed to the 4 originally shown.

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

10. Page 6-64, Figure 6-6

Issue: The figure for Valid Bits Values incorrectly shows bit positions 0-8 as the location of the valid bits. The corrected figure shows the valid bits in positions 0-4.

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

11. Page 6-116

Issue: A row has been added to Table 6-10 describing the sysctl 0x4 type field.

Message	Type	Field 1	Field 2	Field 3	Field 4
Load Control Register	0x4	Register Group Number	N/U	N/U	N/U

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

12. Page 8-6, Figure 6-6

Issue: These first few sentences in the description for system-call entry (type 102) originally read:

> “Provides a procedure number in the system procedure table. This entry must have an entry type of 10_2 and a value in the second word of 0000 027FH. Using this entry, the processor invokes the specified fault handling procedure by means of an implicit call-system operation similar to that performed for the calls instruction. A fault handling procedure in the system procedure table can be called with a system-local call or a system-supervisor call, depending on the entry type in the system-procedure table.”

The corrected text reads:

> “Provides a procedure number in the system procedure table. This entry must have an entry type of 10_2 and a value in the second word of 0000 027FH. The processor computes the system procedure number by shifting right the first word of the fault entry by two bit positions. Using this system procedure number, the processor invokes the specified fault handling procedure by means of an implicit call-system operation similar to that performed for the calls instruction.”

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

13. Page 11-19

Issue: Code was inadvertently omitted from previous revision of the reference document.

The added code reads as follows with original:

```
# Clear level-detect interrupts before return from handler
    ld  INTR_SRC, g0   # Dismiss the extern. interrupt
    lda IPND_MMR, g1   # g1 = IPND MMR address
    lda 0x80, g2       # g2 = mask to clear XINT7 IPND bit

# Loop until IPND bit 7 clears
wait:
    mov 0,g3
    # Try to clear the XINT7 IPND bit
    atmodg1, g2, g3
    bbs 0x7, g3, wait  # Branch until IPND bit 7 clears

# Optionally restore IMSK
    mov r3, IMSK

    ret                # Return from handler
```

Affected Docs: The *i960® Hx Microprocessor User's Manual,* November 1995, release 001.

14. Page 12-15

Issue: The last sentence of the first paragraph originally read: For application debugging with the GMU, conditional branches to regions protected by the GMU should always be predicted taken. The corrected text reads: For application debugging with the GMU, conditional branches to regions protected by the GMU should always be predicted as not taken.

Affected Docs: The *i960® Hx Microprocessor User's Manual,* November 1995, release 001.

15. Page 13-4, Figure 13-2

Issue: The text that appeared near the top center of the diagram:

"V_{CC} and CLKIN Stable to Outputs Valid, maximum 32 CLKIN Periods"

has been deleted.

Affected Docs: The *i960® Hx Microprocessor User's Manual,* November 1995, release 001.

16. Page 13-9, Section 13.2.2.5

Issue: The following sentence has been added to the beginning of the first paragraph:

"When the processor fails the self test, the FAIL# pin asserts and the processor signals the cause of the failure."

Affected Docs: The *i960® Hx Microprocessor User's Manual,* November 1995, release 001.

17. Page 13-11, Section 13.2.2.5

Issue: The address given in the paragraph after the bulleted list has been changed from:

"FEFF FF60H" to

"FF00 0000H"

Affected Docs: The *i960® Hx Microprocessor User's Manual,* November 1995, release 001.

18. Page 13-23, Table 13-7

Issue: The entry for address 64H was originally listed as “Breakpoint Control (BPCON)” -- The entry should be listed as “Reserved (Initialize to Zero).”

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

19. Page 13-37, Figure 13-9

Issue: The text near the top of the figure read: 100 Ohms.

The corrected text reads: 100 Ohms (±5%, 1/8 W)

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

20. Page 13-37, Figure 13-10

Issue: The text near the top of the figure read: 3.3 V V_{CC}.

The corrected text reads: 5 V V_{CC}.

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

21. Page 15-15

Issue: The second sentence originally read: Two short word burst accesses always begin on an even short word boundary (A1=0). The corrected text reads: Two short word burst accesses always begin on a four word boundary (A2=0, A1=0).

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

22. Pages E-41, E-44, E-45, *Examples E-1, E-3, E-4*

Issue: The `cmpinco` instructions originally read: `cmpinco g0, g3, g0`

The corrected text reads: `cmpinco g0, g3, g3`.

Affected Docs: The *i960® Hx Microprocessor User’s Manual*, November 1995, release 001.

23. AP-506, Page 17, “BSTALL” Section, 4th Paragraph

Issue: The BSTALL signal does not always coincide with the BREQ signal. Logically “AND”-ing these two signals can cause an external bus arbiter to ignore a processor stall condition.

Replace the first sentence in the paragraph with the following:

“If BSTALL is used for bus arbitration in an 80960Hx-ready system, the recommendation is to logically “OR” BSTALL and BREQ to indicate when the microprocessor requires the bus.”

Affected Docs: *AP-506: Designing for 80960Cx and 80960Hx Compatibility*, order number 272556.

24. Datasheet, Page 8, Table 6

Issue: BREQ does not indicate whether or not the processor is stalled.

Replace the BREQ pin description with the following.

BREQ	**O**
	H(Q)
	B(Q)
	R(0)
		BUS REQUEST indicates that a bus request is pending in the bus controller. BREQ does not indicate whether or not the processor is stalled. See BSTALL for processor stall status. BREQ can be used with BSTALL to indicate to an external bus arbiter the processor’s bus ownership requirements.

Affected Docs: The *80960HA/HD/HT 32-Bit High-Performance Superscalar Processor* datasheet, August 1997, release 006 and all earlier releases.