80960HA/HD/HT SPECIFICATION UPDATE

Release Date: January, 1997
Order Number: 272830-002

The 80960HA/HD/HT may contain design defects or errors known as errata. Characterized errata that may cause the 80960HA/HD/HT'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

REVISION HISTORY

The errata and information that has changed in this Specification Update since the last revision are outlined below:

REVISION HISTORY (Sheet 1 of 2)

Rev. Date	Version	Description
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12/08/95	1.01	Add errata item #22 Instruction Breakpoints Are Superseded by Invalid Opcode Faults

REVISION HISTORY (Sheet 2 of 2)

Rev. Date	Version	Description
11/17/95	1.00	The A-1 stepping fixes the following errata from the A-0 (A-0 Errata Sheet numbering shown): #6 RESET Has Priority Over HOLD, #9 Data Cache Global Disable Bit (CCON.dci, sf2) May Take 1 Extra Clock Cycle to Complete, #16 Low Temperature Operating Limit is Increased to 25C, and #18 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 #21 Invalidating the Data Cache Automatically Re-enables It Modifications since rev 0.11 (10/27/95) of the A-0 errata sheet... #8 Hold Vcc Above 3.15V and Below 3.45V #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 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

Contact your Intel sales representative to receive copies of the following documents:

Title	Order #
80960HA/HD/HT 32-Bit High Performance Superscalar Processor Data Sheet	272495-004
i960® Hx Microprocessor User's Manual	272484-001
i960® Hx Microprocessor Instruction Set and Register Quick Reference	272792-001
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 80960HA/HD/HT 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 (Sheet 1 of 2)

No.	Steppings					Page	Status	ERRATA
	A0	A1	A2	B0	B1			
1.	X	X	X			14	Fixed	Parity Failure on 8- and 16-bit Unaligned Loads
2.	X	X	X	X	X	14	NoFix	Read Wrong Location from Non-Burst, 8- and 16-bit Memory Regions
3.	X	X	X	X	X	15	NoFix	Breakpoints on Stacks Produce Wrong Fault IP
4.	X	X	X			15	Fixed	Parity Faults May Not Report Correct Address and Access Type
5.	X	X	X			16	Fixed	PMCON15 Temporarily Initialized Incorrectly During RESET
6.	X	X	X			16	Fixed	BCON Register is not Cleared Before Software Reset
7.	X	X	X			17	Fixed	MODTC Command Can Set TC Register Event Flags
8.	X	X	X	X	X	17	NoFix	Parity Faults Cannot Be Disabled Separate from the PCHK# Pin
9.	X	X	X			17	Fixed	Timer Terminal Count (TMR.tc) Bit Cannot Bear Polling
10.	X	X	X	X	X	18	NoFix	Return Instruction Pointer (RIP) Cannot be Stored by Software
11.	X	X	X			19	Fixed	WAIT# Pin Asserts During NXDA Wait States
12.	X	X	X			19	Fixed	Software Interrupts Can Access the Wrong Handler Address
13.	X	X	X			20	Fixed	Invalidating the Data Cache Automatically Re-enables It
14.	X					21	Fixed	RESET Has Priority Over HOLD
15.	X					21	Fixed	Data Cache Global Disable Bit (CCON.dci, sf2) May Take 1 Extra Clock Cycle To Complete
16.	X					22	Fixed	Low Temperature Operating Limit Increased to 25°C
17.	X					22	Fixed	IPND Register Not Cleared Automatically

Errata (Sheet 2 of 2)

No.	Steppings					Page	Status	ERRATA
	A0	A1	A2	B0	B1			
18.	X	X	X			23	Fixed	Cycle Type Bits (CT3:0) Do Not Indicate Some Fault Types
19.	X	X	X	X	X	25	NoFix	Operation Fault Occurs When Clearing the IMASK (sf1) Register
20.	X	X	X			26	Fixed	Using atmod or sysctl to Change IMSK or IPND MMRS Can Hang the Processor
21.	X	X	X			27	Fixed	Storing the Contents of the I_CACHE to External Memory Also Disables the Cache
22.	X	X	X			28	Fixed	PCHK# Pin Does Not Indicate Parity Failures On HD and HT Processors
23.	X	X	X	X	X	28	NoFix	Spurious INVALID_OPCODE Faults Can Occur with Level-Detect Interrupts
24.	X	X	X	X		32	Fixed	Parity Can Fail on Reliable Data and Can Pass on Corrupted Data

Specification Changes

No.	Document Revision	Page	Status	SPECIFICATION CHANGES
1.	272495-003	33		Hold V_{CC} Above 3.15V and Below 3.45V
2.	272495-003	33		TRST# Input Can Be Tied Low
3.	272495-003	34		Halt Mode Does Not Conserve Power

Specification Clarifications

No.	Document Revision	Page	Status	SPECIFICATION CLARIFICATIONS
1.	272484-001	35		Burst Accesses on 8- and 16-Bit Buses Do Not Behave Like the Cx Processor
2.	272484-001	36		Instruction Breakpoints Are Superseded by Invalid Opcode Faults

Documentation Changes

No.	Document Revision	Page	DOCUMENTATION CHANGES
1.	272484-001	38	Page 3-11, Table 3-4
2.	272484-001	38	Page 3-11, Table 3-4
3.	272484-001	38	Page 3-26
4.	272484-001	38	Page 4-6, Section 4.4.3
5.	272484-001	39	Page 6-45
6.	272484-001	39	Page 6-60, Table 6-8
7.	272484-001	39	Page 6-61, Figure 6-4
8.	272484-001	39	Page 6-62, Table 6-9
9.	272484-001	40	Page 6-63, Figure 6-5
10.	272484-001	40	Page 6-64, Figure 6-6
11.	272484-001	40	Page 6-116
12.	272484-001	40	Page 8-6, Figure 6-6
13.	272484-001	41	Page 11-19
14.	272484-001	41	Page 12-15
15.	272484-001	42	Page 13-4, Figure 13-2
16.	272484-001	42	Page 13-9, Section 13.2.2.5
17.	272484-001	42	Page 13-11, Section 13.2.2.5
18.	272484-001	43	Page 13-23, Table 13-7
19.	272484-001	43	Page 13-37, Figure 13-9
20.	272484-001	43	Page 13-37, Figure 13-10
21.	272484-001	43	Page 15-15
22.	272484-001	43	Pages E-41, E-44, E-45, Examples E-1, E-3, E-4

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:

DEVICE IDENTIFIER CODES FOUND IN THE g0 REGISTER AFTER RESET

	Device		
Stepping	**80960HA**	**80960HD**	**80960HT**
A-0	0x08840013	0x08841013	0x18840013
A-1 & A-2	0x08841013	0x08841013	0x18841013
B-0 & B-1	0x08842013	0x08841013	0x18842013

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

JTAG REGISTERS

See the data sheet, release -004, dated June 1996, Section 5.1 80960Hx Boundary Scan Chain, pages 74-78 for the boundary scan chain definition.

See the data sheet, release -004, dated June 1996, 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: 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 VCC 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 BRANCH CALL RETURN SUPERVISOR MARK/BREAKPOINT breakpoint always works correctly, *mark* when Mark Trace Mode is not set in the TC register.	PRERETURN MARK/BREAKPOINT *mark* when Mark Trace Mode is set in the TC register, *fmark*.
2H Operation	INVALID_OPCODE UNIMPLEMENTED UNALIGNED INVALID_OPERAND non-existent *sfr*, unaligned long-, triple-, or quad-register, undefined register, writing to RIP.	UNIMPLEMENTED *sysctl* message type 04H INVALID_OPERAND undefined *sysctl*, *icctl*, *dcctl*, or *intctl* operand.
3H Arithmetic	INTEGER_OVERFLOW ZERO_DIVIDE	INTEGER_OVERFLOW integer divide overflow (*divi*)

AULT TYPE	CT2 BIT WORKS FOR...	
5H Constraint	RANGE all other cases.	RANGE only if a *fault<cc>* test evaluates true.
7H Protection	BAD_ACCESS GMU detection	BAD_ACCESS GMU protection LENGTH
8H Machine	PARITY_ERROR	
AH Type	MISMATCH execute a privileged instruction while in user mode (*intdis, inten*), 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).	MISMATCH execute a privileged instruction while in User mode (*modpc*, *sysctl*, *icctl*, *dcctl, intctl*).
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:

A.

```
Clear_IMASK_sfr_register
microcoded instruction
```

B.

```
Clear_IPND_sfr_register
microcoded instruction
```

C.

```
Clear_IMSK_sfr_Bit_for_Posted_Interrupt
microcoded instruction
```

D.

```
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 –

a. dead reckoning - calculate the worst case latencies.

b. delay the ISR until the interrupt is dismissed - wait until the dismissing bus access completes before proceeding with the ISR execution.

c. 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 Hx processor must be the only bus master in the system (the HOLD, BOFF# and 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 Hx 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 Hx 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 Hx 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
   ldaIPND_MMR, g1# g1 = IPND MMR address
   lda0x80, g2    # g2 = mask to clear XINT7 IPND bit

# Loop until IPND bit 7 clears
wait:
   mov0,g3
   # Try to clear the XINT7 IPND bit
   atmodg1, g2, g3
   bbs0x7, g3, wait# Branch until IPND bit 7 clears

# Optionally restore IMSK
   movr3, 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 Hx 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).

SPECIFICATION CHANGES

1. Hold V_{CC} Above 3.15V and Below 3.45V

ISSUE: The processor does not operate reliably under all conditions while V_{CC} is outside the range $3.15 \leq V_{CC} \leq 3.45$ V.

Below 3.15 V V_{CC}, the processor core does not operate at high core speeds. This low V_{CC} errata applies only to the 80960HD-66 processor.

Above 3.45 V V_{CC}, signal ringing on some output pins causes AC specification violations. This ringing behavior affects all versions of the processor.

IMPLICATION: Random system failures, faults, and data corruption can occur if V_{CC} is not regulated within these limits.

WORKAROUND: Regulate V_{CC} within the commonly accepted "±5%" tolerance range $3.15 \leq V_{CC} \leq 3.45$ V.

AFFECTED DOCUMENT(S): *80960HA/HD/HT 32-Bit High Performance Superscalar Processor Data Sheet,* May 1995, revision -003.

FROM/TO REFERENCE: Table 17. Operating Conditions, page 27. Replace the first 3 rows with the following:

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage	3.15	3.45	V
V_{CC5}	Input Protection Bias	3.15	5.5	V

2. TRST# Input Can Be Tied Low

ISSUE: The Boundary Scan TRST# pin can be tied low indefinitely, thus resetting the Test Access Port (TAP) logic. This is an improvement over the previously published specs which forbade tying the pin low.

IMPLICATION: Applications that do not use the Boundary Scan features can safely disable those features by grounding the TRST# input pin.

IEEE 1149.1 Boundary Scan compliance is not affected either way by this change, albeit this change does make the TAP controller easier to reset as described in the *IEEE Standard Test Access Port and Boundary-Scan Architecture* specification describing IEEE 1149.1, section 3.6.2.

WORKAROUND: None required.

AFFECTED DOCUMENT(S): *80960HA/HD/HT 32-Bit High Performance Superscalar Processor Data Sheet,* May 1995, revision -003. This is fixed in the current data sheet revision -004, dated June 1996, which is now available from Intel Literature Center.

FROM/TO REFERENCE: Table 6. 80960Hx Processor Family Pin Descriptions, page 10, middle row labeled "TRST#". Replace the descriptions with the following:

> "**TEST RESET** asynchronously resets the Test Access Port (TAP) controller. TRST# must be held low at least 10,000 clock cycles after power-up. One method is to provide TRST# with a separate power-on-reset circuit. TRST# includes a built-in pull-up resistor. (See Table 19. DC Characteristics for the value.) Pull this pin low when not using the TAP controller."

3. Halt Mode Does Not Conserve Power

ISSUE: Halt mode does not provide nearly expected power savings, so it is being de-featured.

Power consumption in Halt mode depends heavily on instructions executed and processor state just prior to the Halt instruction. Since Halt mode does not stop on-chip clocks, the processor continues to spin in the last state before Halting. Depending on power consumption of the particular instruction and state, the processor can dissipate more than the typical power specified in the data sheet.

Beginning with the B-0 stepping, the processor does not recognize the ***halt*** instruction; it produces an INVALID_OPCODE fault.

IMPLICATION: The 80960Hx processor does not have a low power idle mode.

WORKAROUND: Do not use the Halt instruction. Workaround options are described in the Tech Bit "*Reduced Power Options for the 80960HA/HD/HT*."

AFFECTED DOCUMENT(S): *80960HA/HD/HT 32-Bit High Performance Superscalar Processor Data Sheet,* May 1995, revision -003. This is fixed in the current data sheet revision -004, dated June 1996, which is now available from Intel Literature Center.

FROM/TO REFERENCE: Table 4. 80960Hx Instruction Set, page 4, bottom column labeled "Processor Mgmt". Delete "Halt[1,2]" from the list.

Table 5. Pin Description Nomenclature, page 5, bottom row labeled "P(...)". Delete row.

Table 6. 80960Hx Processor Family Pin Descriptions, pages 6-10, column labeled "Type". Delete every instance of "P(...)".

Table 6. 80960Hx Processor Family Pin Descriptions, page 9, second row labeled "CT3:0". Replace the following text:

> "Processor in HALT mode" with... "Reserved for future products"

Table 19. DC Characteristics, page 29, row labeled "I_{HALT}". Delete the row.

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 Hx 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					Hx 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 Hx 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 (Sheet 1 of 2)

	Word 0					Word 1					Hx Accesses
Access beginning on...	0	1	2	3		4	5	6	7		
... Byte 0 (aligned)											Burst 2 shorts
... Byte 1											Burst byte & short, byte

16-Bit Bus Behavior for a Word Access (Sheet 2 of 2)

... 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 DOCUMENT(S): 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 DOCUMENT(S): 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."

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. The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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"

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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. The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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."

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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;
```

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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". The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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. The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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. The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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.

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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.

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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."

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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
   ldaIPND_MMR, g1# g1 = IPND MMR address
   lda0x80, g2    # g2 = mask to clear XINT7 IPND bit

# Loop until IPND bit 7 clears
wait:
   mov0,g3
   # Try to clear the XINT7 IPND bit
   atmodg1, g2, g3
   bbs0x7, g3, wait# Branch until IPND bit 7 clears

# Optionally restore IMSK
   movr3, IMSK

   ret            # Return from handler
```

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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.

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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.

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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."

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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"

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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)."

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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)

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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}.

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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).

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): 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`.

The corrected page is appended to this document.

AFFECTED DOCUMENT(S): The *i960® Hx Microprocessor User's Manual,* November 1995, release 001.

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

Register Name	Memory-Mapped Address	Access Type
(IPB1) Instruction Address Breakpoint Register 1	FF00 8404H	Sysctl- RwG/WwG
(IPB2) Instruction Address Breakpoint Register 2	FF00 8408H	Sysctl- RwG/WwG
(IPB3) Instruction Address Breakpoint Register 3	FF00 840CH	Sysctl- RwG/WwG
(IPB4) Instruction Address Breakpoint Register 4	FF00 8410H	Sysctl- RwG/WwG
(IPB5) Instruction Address Breakpoint Register 5	FF00 8414H	Sysctl- RwG/WwG
Reserved	FF00 8418H 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
(DAB2) Data Address Breakpoint Register 2	FF00 8428H	R/W, WwG
(DAB3) Data Address Breakpoint Register 3	FF00 842CH	R/W, WwG
(DAB4) Data Address Breakpoint Register 4	FF00 8430H	R/W, WwG
(DAB5) Data Address Breakpoint Register 5	FF00 8434H	R/W, WwG
Reserved	FF00 8438H to FF00 843FH	—
(BPCON) Breakpoint Control Register	FF00 8440H	WwG
(XBPCON) Extended Breakpoint Control Register	FF00 8444H	WwG
Reserved	FF00 8448H to FF00 84FFH	—
Interrupts		
(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	—

NOTE:
Shaded rows indicate reserved areas.

3

Page 3-11, Table 3-4

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

R/W, WwG

The corrected entries should read:

WwG

Page 3-11, Table 3-4

The incorrect 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.

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

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

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

Software should not use **modpc** to modify execution mode or trace fault state flags except under special circumstances, such as in initialization code. Normally, execution mode is changed through the call and return mechanism. See section 6.2.43, “modpc” (pg. 6-80) for more details.

Page 3-26, Second Paragraph

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 that are used to enable specific tracing modes and record trace events, respectively. Trace controls are described in CHAPTER 9, 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 that 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 access to system debugging software or a system monitor, even if an application’s program destroys its own stack.
- When an instruction executed in supervisor mode causes a bus access to occur, the processor asserts an external supervisor pin ($\overline{SUP}$) for loads, stores and instruction fetches. Hardware protection of system code or data can be implemented by using the supervisor pin to qualify write accesses to the protected memory.

4.4.3 Loading and Locking Instructions in the Instruction Cache

The processor can be directed to load a block of instructions into the cache and then lock out all normal updates to the cache. This cache load-and-lock mechanism is provided to minimize latency on program control transfers to key operations such as interrupt service routines. The block size that can be loaded and locked on the i960 Hx processor is any multiple of 4-Kbytes up to the full 16-Kbyte capacity of the cache. Any code can be locked, not just interrupt routines.

An **icctl** instruction invokes the load-and-lock mechanism for one, two, three, or all four 4-Kbyte ways of the instruction cache. Legacy software from the i960 Cx processor can still use the **sysctl** instruction to lock the cache, but with reduced flexibility. **sysctl** can load and lock only one 4-Kbyte way of the instruction cache due to backwards compatibility with the i960 Cx processor definition of the **sysctl** instruction. New software for the i960 Hx processor should use **icctl** for all instruction cache manipulations. With either instruction, when the lock option is selected, the processor loads the cache starting at an address specified as an operand to the instruction.

Page 4-6, Section 4.4.3

This sentence has been added to the end of the first paragraph:

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

4.4.4 Instruction Cache Visibility

Instruction cache status can be determined by issuing **icctl** with an instruction-cache status message. To facilitate debugging, the instruction cache contents, instructions, tags and valid bits can be written to memory. This is done by issuing **icctl** with the store cache operation.

4.4.5 Instruction Cache Coherency

The i960 Hx processor does not snoop the bus to prevent instruction cache incoherency. The cache does not detect modification to program memory by loads, stores or actions of other bus masters. Several situations may require program memory modification, such as uploading code at initialization or loading from a backplane bus or a disk drive.

The application program is responsible for synchronizing its own code modification and cache invalidation. In general, a program must ensure that modified code space is not accessed until modification and cache-invalidate are completed. To achieve cache coherency, instruction cache contents should be invalidated after code modification is complete. The **icctl** instruction invalidates the instruction cache for the i960 Hx processor. Alternately, i960 Cx processor legacy software can use the **sysctl** instruction.

4.4.6 Instruction Cache Interaction with Guarded Memory

The Guarded Memory Unit (GMU) protects only external memory accesses. Instructions executed from the instruction cache will not activate the GMU mechanisms since these instructions do not cause an external instruction fetch.

```
case 6:     # Store data cache sets to memory pointed to by src2.
                start = src_dst[15:0]           #   Starting set number.
                end   = src_dst[31:16]          #   Ending set number.
                                                #   (zero-origin).
                if (end >= Dcache_max_sets) end = Dcache_max_sets - 1;
                if (start > end) generate_fault
                                        (OPERATION.INVALID_OPERAND);
                memadr = src2;                  #   Must be word-aligned.
                if (0x3 & memadr! = 0)
                generate_fault(OPERATION.INVALID_OPERAND)
                for (set = start; set <= end; set++){
                    # Set_Data is described at end of this code flow.
                     memory[memadr] = Set_Data[set];
                     memadr += 4;
                     for (way = 0; way < numb_ways; way++)
                         {memory[memadr] = tags[set][way];
                          memadr += 4;
                          memory[memadr] = valid_bits[set][way];
                          memadr += 4;
                         for (word = 0; word < words_in_line; word++)
                              {memory[memadr] =
                                          Dcache_line[set][way][word];
                               memadr += 4;
                              }
                              }
                }
                break;
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;
default:    # Reserved.
                generate_fault(OPERATION.INVALID_OPERAND);
                break;
    }
order_wrt(subsequent_operations)
```

6

Page 6-45, Case 8

Added Case 8. It was inadvertently omitted from previous revision.

Faults:

STANDARD	Refer to section 6.1.6, “Faults” (pg. 6-5).
TYPE.MISMATCH	Attempt to execute instruction while not in supervisor mode.
OPERATION.INVALID_OPERAND	

6.2.33 icctl

Mnemonic: **icctl** Instruction-cache Control

Format:

icctl	*src1*,	*src2*,	*src/dst*
	reg/lit/sfr	reg/lit/sfr	reg/sfr

Description: Performs management and control of the instruction cache including disabling, enabling, invalidating, loading and locking, getting status, and storing cache sets to memory. Operations are indicated by the value of *src1*. Some operations also use *src2* and *src/dst*. When needed by the operation, the processor orders the effects of the operation with previous and subsequent operations to ensure correct behavior. For specific function setup, see the following tables and diagrams:

Page 6-60, Table 6-8

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

Table 6-8. icctl Operand Fields

Function	*src1*	*src2*	*src/dst*
Disable I-cache	0	NA	NA
Enable I-cache	1	NA	NA
Invalidate I-cache	2	NA	NA
Load and lock I-cache	3	*src*: Starting address of code to lock.	Number of ways to lock.
Get I-cache status	4	NA	*dst:* Receives status (see Figure 6-4).
Get I-cache locking status	5	NA	*dst:* Receives status (see Figure 6-4)
Store I-cache sets to memory	6	Destination address for cache sets	*src*: I-cache set #'s to be stored (see Figure 6-4).

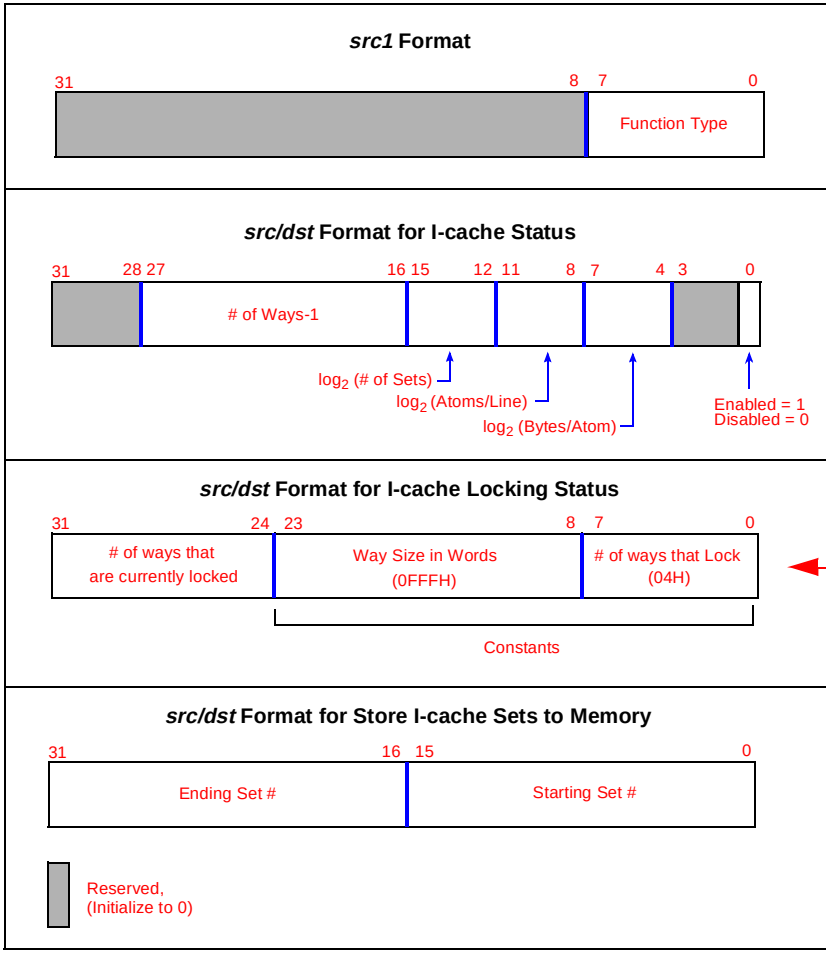


Figure 6-4. icctl *src1* and *src/dst* Formats

Page 6-61, Figure 6-4

In Figure 6-4 under "Src/Dst Format for I-cache Locking Status" constant fixed values for bits 0 - 23 are shown.

Table 6-9. icctl Status Values and Instruction Cache Parameters

Value	Value on 80960Hx
bytes per atom	4
atoms per line	8
number of sets	128
number of ways	4
cache size	16-Kbytes
Status[0] (enable/disable)	0 or 1
Status[1:3] (reserved)	0
Status[7:4] (log2(bytes per atom))	2
Status[11:8] (log2(atoms per line))	3
Status[15:12] (log2(number of sets))	8
Status[27:16] (number of ways - 1)	3
Lock Status[7:0] (number of blocks that lock)	
Lock Status[23:8] (block size in words)	1024
Lock Status[31:24] (number of blocks that are locked)	0-4

Page 6-62, Table 6-9

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

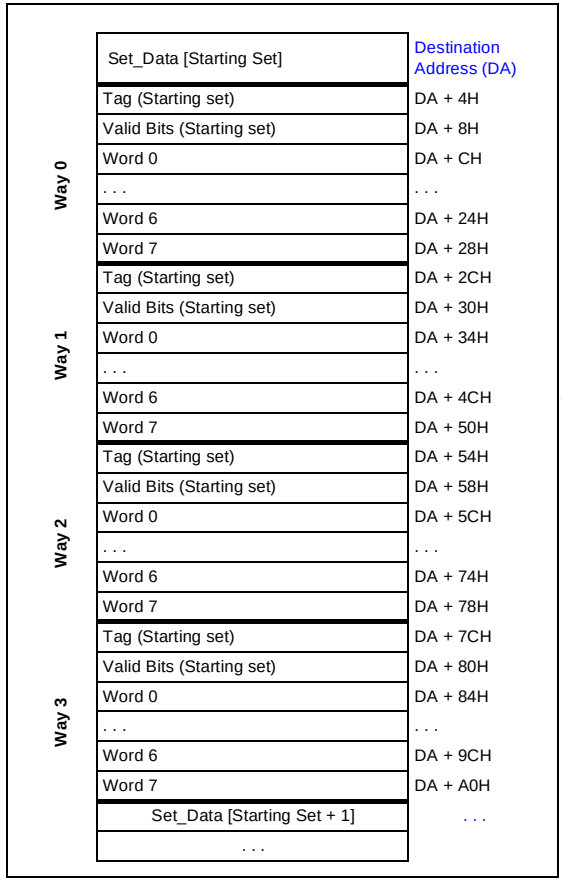
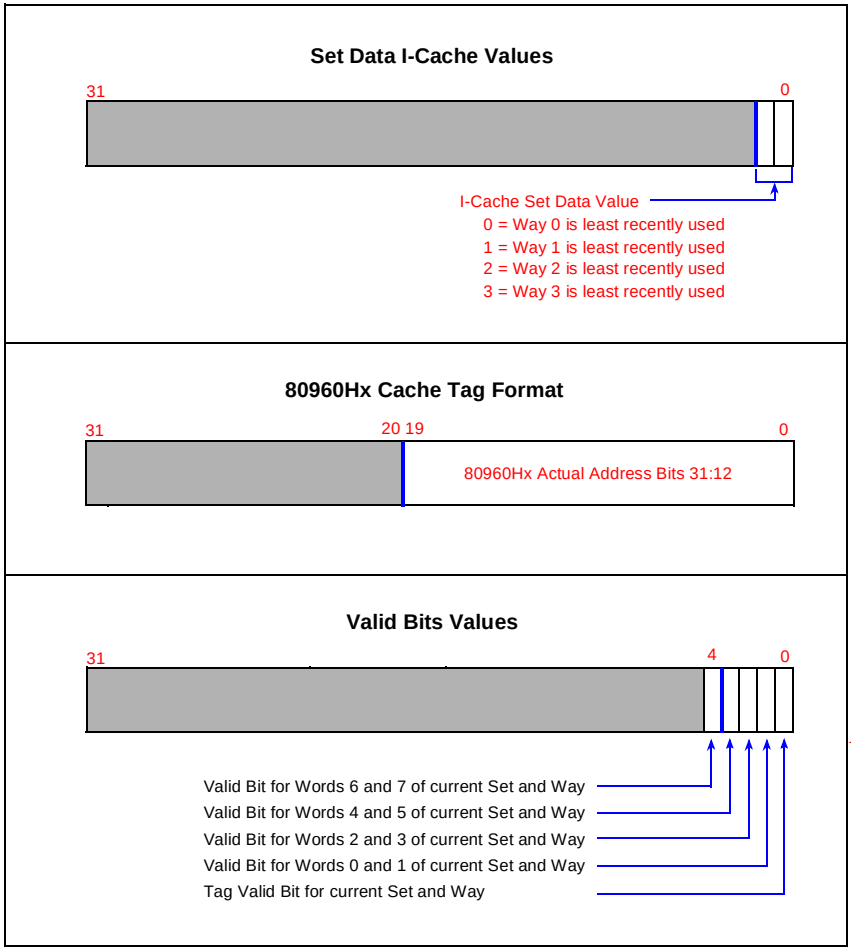


Figure 6-5. Store Instruction Cache to Memory Output Format

Page 6-63, Figure 6-5

Each way in Figure 6-5 should have 8 words, as opposed to the 4 originally shown.



Page 6-64, Figure 6-6

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.

Figure 6-6. I-Cache Set Data, Tag and Valid Bit Formats

6.2.67 sysctl

Mnemonic: **sysctl** System Control

Format:

sysctl	*src1*, reg/lit/sfr	*src2*, reg/lit/sfr	*src/dst* reg/sfr

Description: Performs system management and control operations including requesting software interrupts, invalidating the instruction cache, configuring the instruction cache, processor reinitialization, modifying memory-mapped registers, and acquiring breakpoint resource information.

Processor control function specified by the message field of *src1* is executed. The type field of *src1* is interpreted depending upon the command. Remaining *src1* bits are reserved. The *src2* and *src3* operands are also interpreted depending upon the command.

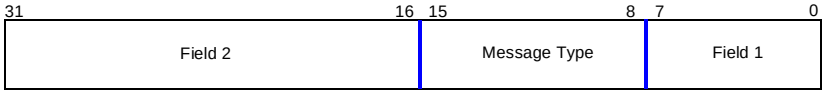


Figure 6-7. Src1 Operand Interpretation

Table 6-10. sysctl Field Definitions

Message	*src1*			*src2*	*src/dst*
	Type	Field 1	Field 2	Field 3	Field 4
Request Interrupt	0x0	Vector Number	N/U	N/U	N/U
Invalidate Cache	0x1	N/U	N/U	N/U	N/U
Configure Instruction Cache	0x2	Cache Mode Configuration (See Table 6-11)	N/U	Cache load address	N/U
Reinitialize	0x3	N/U	N/U	Starting IP	PRCB Pointer
Load Control Register	0x4	Register Group Number	N/U	N/U	N/U
Modify Memory-Mapped Control Register (MMR)	0x5	N/U	Lower 2 bytes of MMR address	Value to write	Mask
Breakpoint Resource Request	0x6	N/U	N/U	N/U	Breakpoint info (See Figure 6-8)

NOTE: Sources and fields that are not used (designated N/U) are ignored.

Page 6-116, Table 6-10

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

As indicated in Figure 8-2, two fault table entry types are allowed: local-call entry and system-call entry. Each is two words in length. The entry type field (bits 0 and 1 of the entry's first word) and the value in the entry's second word determine the entry type.

local-call entry (type 00_2) — Provides an instruction pointer for the fault handling procedure. The processor uses this entry to invoke the specified procedure by means of an implicit local-call operation. The second word of a local procedure entry is reserved. It must be set to zero when the fault table is created and not accessed after that.

system-call entry (type 10_2) — Provides a procedure number in the system procedure table. This entry must have an entry type of 102 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.

Other entry types (01_2 and 11_2) are reserved and have unpredictable behavior.

To summarize, a fault handling procedure can be invoked through the fault table in any of three ways: a local call, a system-local call or a system-supervisor call.

8.4 STACK USED IN FAULT HANDLING

The i960 architecture does not define a dedicated fault handling stack. Instead, to handle a fault, the processor uses either the user, interrupt or supervisor stack, whichever is active when the fault is generated. There is, however, one exception: if the user stack is active when a fault is generated and the fault handling procedure is called with an implicit system supervisor call, the processor switches to the supervisor stack to handle the fault.

8.5 FAULT RECORD

When a fault occurs, the processor records information about the fault in a fault record in memory. The fault handling procedure uses the information in the fault record to correct or recover from the fault condition and, if possible, resume program execution. The fault record is stored on the same stack that the fault handling procedure will use to handle the fault.

Page 8-6, System-Call Entry

These first few sentences in the description for system-call entry (type 10_2) 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.

11.7.2 Interrupt Detection Options

The $\overline{\text{XINT7:0}}$ pins can be programmed for level-low or falling-edge detection when used as dedicated inputs. All dedicated inputs plus the $\overline{\text{NMI}}$ pin are programmed (globally) for fast sampling or debounce sampling. Expanded-mode inputs are always sampled in debounce mode. Pin detection and sampling options are selected by programming the ICON register.

When falling-edge detection is enabled and a high-to-low transition is detected, the processor sets the corresponding pending bit in the IPND register. The processor clears the IPND bit upon entry into the interrupt handler.

When a pin is programmed for low-level detection, the pin's bit in the IPND register remains set as long as the pin is asserted (low). The processor attempts to clear the IPND bit on entry into the interrupt handler; however, if the active level on the pin is not removed at this time, the bit in the IPND register remains set until the source of the interrupt is deactivated and the IPND bit is explicitly cleared by software. Software may attempt to clear an interrupt pending bit before the active level on the corresponding pin is removed. In this case, the active level on the interrupt pin causes the pending bit to remain asserted.

After the interrupt signal is deasserted, the handler then clears the interrupt pending bit for that source before return from handler is executed. If the pending bit is not cleared, the interrupt is re-entered after the return is executed.

Example 11-5 demonstrates how a level detect interrupt is typically handled. The example assumes that the **ld** from address "timer_0," deactivates the interrupt input.

Example 11-5. Return from a Level-detect Interrupt

```
# 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
```

The debounce sampling mode provides a built-in filter for noisy or slow-falling inputs. The debounce sampling mode requires that a low level is stable for three consecutive cycles before the expanded mode vector is resolved internally. Expanded mode interrupts are always sampled using the debounce sampling mode. This allows for skew time between changing outputs of external priority encoders.

Page 11-19, Example 11-5

Cleaned up polling code and revised.

Due to instruction prefetching, a spurious PROTECTION.BAD_ACCESS fault may be generated when the target of a branch is in a region fetch-protected by the GMU and the branch is predicted to be taken, but not actually taken. For application debugging with the GMU, conditional branches to regions protected by the GMU should always be predicted as not taken.

Software should not program the GMU to protect the memory-mapped registers in the range of addresses FFFFFF00H through FFFFFFFFH as this can lead to the unexpected generation of PROTECTION.BAD_ACCESS faults.

In general, the Interrupt Table should not be protected against Supervisor mode accesses. Protecting the Interrupt Table from Supervisor mode writes is acceptable if it can be guaranteed that no software posting of interrupts will occur. Protecting the Interrupt Table against Supervisor mode reads will cause trouble if any hardware interrupts, including the NMI, occur. Violation of these cautions will result in improper system behavior.

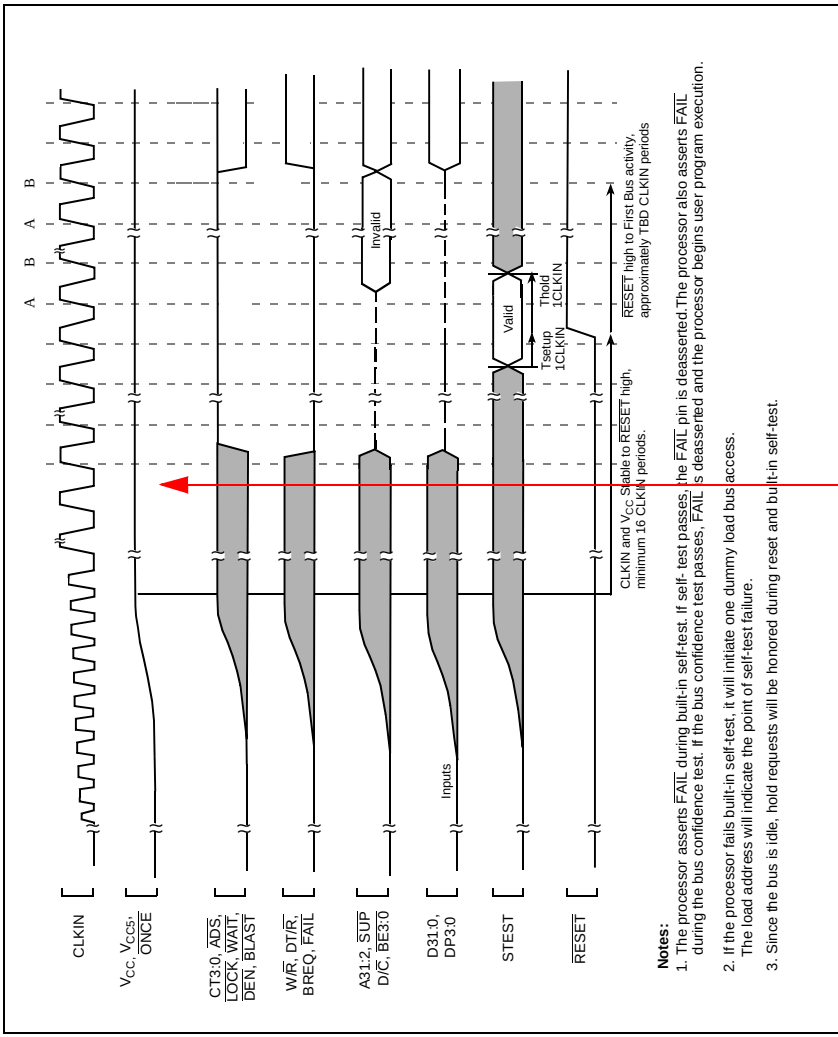
Page 12-15, First Paragraph

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.



Notes:
1. The processor asserts FAIL during built-in self-test. If self- test passes, the FAIL pin is deasserted.The processor also asserts FAIL during the bus confidence test. If the bus confidence test passes, FAIL is deasserted and the processor begins user program execution.
2. If the processor fails built-in self-test, it will initiate one dummy load bus access. The load address will indicate the point of self-test failure.
3. Since the bus is idle, hold requests will be honored during reset and built-in self-test.

Figure 13-2. Cold Reset Waveform

Page 13-04, Figure 13-2

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.

When the processor detects a System Error, it asserts the $\overline{\text{FAIL}}$ pin, drives a fail code message onto the address bus, and stops execution at the point of failure. The only way to resume normal operation of the processor is to perform a reset operation. Because System Error generation can occur after the bus confidence test and even after initialization during normal processor operation, the $\overline{\text{FAIL}}$ pin is a logical "1" before the detection of a System Error.

13.2.2.5 Self Test Failure Codes

When the processor fails the self test, the $\overline{\text{FAIL}}$ pin asserts and the processor signals the cause of the failure. The processor uses only one read bus transaction to signal the fail code message; the address of the bus transaction is the fail code itself. The fail code is of the form: **0xfeffff***nn*; bits 6 to 0 contain a mask recording the possible failures. Bit 7, when 1, indicates the mask contains failures from BIST; when 0, the mask indicates other failures. The fail codes are shown in Table 13-3 and
Table 13-4.

Page 13-9, Section 13.2.2.5

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."

Table 13-3. Fail Codes for BIST (bit 7 = 1)

Bit	When set:
6	On-chip Data-RAM failure detected by BIST
5	Internal Microcode ROM failure detected by BIST
4	I-cache failure detected by BIST
3	D-cache failure detected by BIST
2	Local-register cache or processor core failure detected by BIST
1	Always zero.
0	Always zero.

Table 13-4. Remaining Fail Codes (bit 7 = 0)

Bit	When set:
6	Always One; this bit does not indicate a failure.
5	Always One; this bit does not indicate a failure.

Several data structures are typically included as part of the IMI because values in these data structures are accessed by the processor during initialization. These data structures are usually programmed in the systems's boot ROM, located in memory region 15 of the address space.

The required data structures are:

- PRCB
- IBR
- System procedure table
- Control table
- Interrupt table
- Fault table

To ensure proper processor operation, the PRCB, system procedure table, control table, interrupt table, and fault table must not be located in architecturally reserved memory — addresses reserved for on-chip Data RAM and addresses at and above FF00 0000H. In addition, each of these structures must start at a word-aligned address; a System Error occurs if any of these structures are not word-aligned (see section 13.2.2.3).

At initialization, the processor loads the Supervisor Stack Pointer (SSP) from the system procedure table, aligns it to a 16-byte boundary, and caches the pointer in the SSP memory-mapped control register (see section 3.3, "MEMORY-MAPPED CONTROL REGISTERS (MMRs)" (pg. 3-6)). Recall that the Supervisor Stack Pointer is located in the preamble of the system procedure table at byte offset 12 from the base address. The system procedure table base address is programmed in the PRCB. Consult section 7.5.1, "System Procedure Table" (pg. 7-16) for the format of this table.

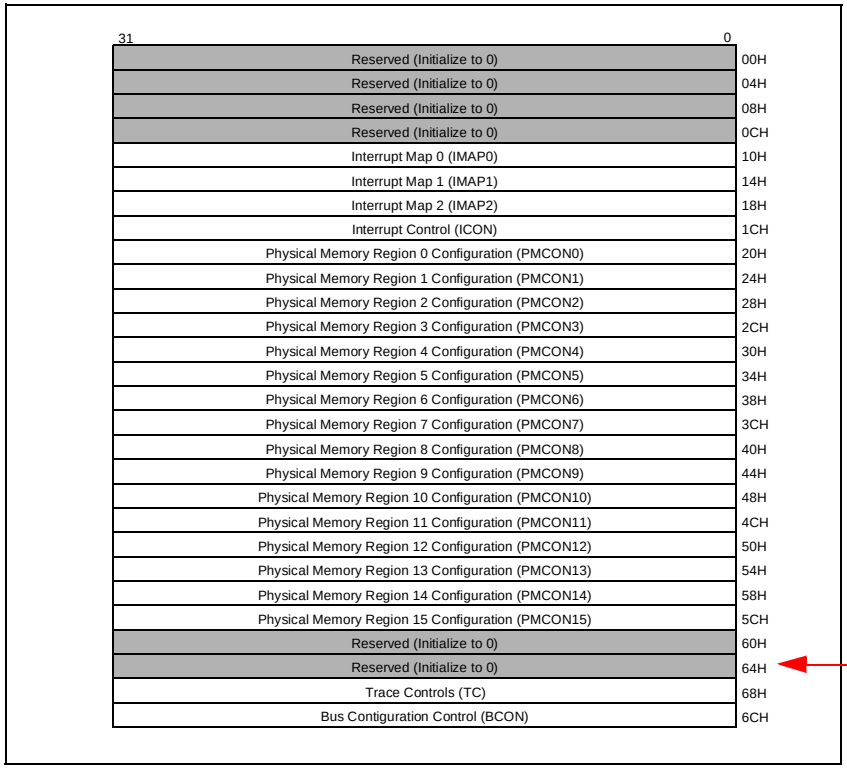
At initialization, the NMI vector loads from the interrupt table into location 0000 0000H of the internal data RAM. The interrupt table is typically programmed in the boot ROM and then relocated to internal RAM by reinitializing the processor.

Typically, applications locate the fault table in boot ROM. To locate the fault table in RAM, the processor must be reinitialized.

The remaining data structures that an application may need are the user stack, supervisor stack and interrupt stack. Applications must locate these stacks in a system's RAM.

Page 13-11,Third Paragraph

The address given in the paragraph after the bulleted list has been changed from FEFF FF60H to FF00 0000H.



Page 13-23, Figure 13-7

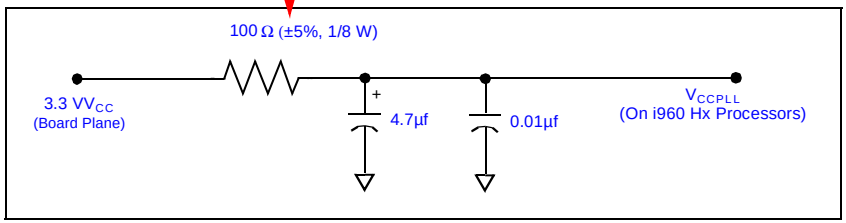
The entry for address 64H was originally listed as "Breakpoint Control (BPCON)"

The entry should be listed as "Reserved (Initialize to Zero)".

Figure 13-7. Control Table

13.4 DEVICE IDENTIFICATION ON RESET

The DEVICEID memory-mapped register contains a number characterizing the microprocessor type and stepping. During initialization, the processor places the DEVICEID register value into g0.



Page 13-37, Figure 13-9

The text near at the top of the figure read:

100Ω

The corrected text reads:

100Ω (±5%, 1/8 W)

Figure 13-9. V_{CCPLL} Lowpass Filter

13.6.3 VCC5 PIN REQUIREMENTS

In mixed voltage systems that drive the i960 Hx processor inputs in excess of 3.3V, the V_{CC5} pin must be connected to the system's 5V supply. To limit current flow into the V_{CC5} pin, there is a limit to the voltage differential between the V_{CC5} pin and the other V_{CC} pins. The voltage differential (V_{DIFF})between the 80960Hx V_{CC5} pin and its 3.3V V_{CC} pins should never exceed 2.25V. This limit applies to power up, power down, and steady-state operation. See the *80960HA/HD/HT Embedded 32-bit Microprocessor Data Sheet* for more details.

If the voltage difference requirements cannot be met due to system design limitations, an alternate solution may be employed. As shown in Figure 13-10, a minimum of 100 ohm series resistor may be used to limit the current into the V_{CC5} pin. This resistor ensures that current drawn by the V_{CC5} pin does not exceed the maximum rating for this pin.

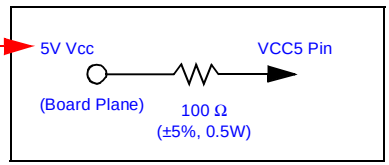


Figure 13-10. V_{CC5} Current-Limiting Resistor

This resistor is not necessary in systems that can guarantee the V_{DIFF} specification. In 3.3V-only systems and systems that drive the i960 Hx processor pins from 3.3V logic, connect the V_{CC5} pin directly to the 3.3V V_{CC} plane.

Page 13-37, Figure 13-10

The text near at the top of the figure read:

3.3V Vcc

The corrected text reads:

5 V Vcc

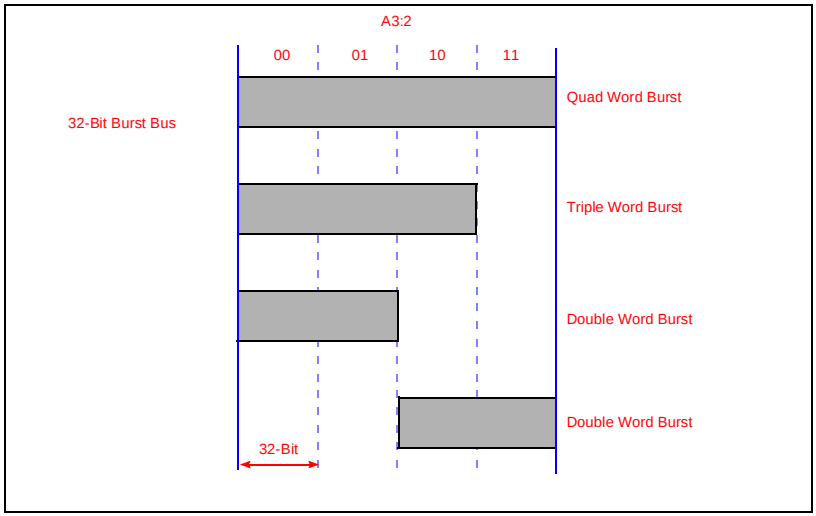


Figure 15-5. 32-Bit-Wide Data Bus Bursts

Burst accesses for a 16-bit bus are always aligned to even short word boundaries. A four short word burst access always begins on a four short word boundary (A2=0, A1=0). Two short word burst accesses always begin on a four word boundary (A2=0, A1=0). Single short word transfers occur on single short word boundaries (see Figure 15-6). For a 16-bit bus, valid data is transferred on data pins D15:0. Upper data lines D31:16 are also driven on writes. For aligned accesses, the values are a duplicate of those driven on D15:0.

Page 15-15, Figure 15-5

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).

E.2.7.1 Loads and Stores

Separate load instructions from instructions that use load data. Remember that store instructions can also be reordered. Although it returns no results to a register, a poorly placed store in front of a critical load slows down the load. Reorder to issue the load first. Example E-1 shows a simple change that saved one clock from a five-clock loop.

Example E-1. Overlapping Loads (Checksum)

```
loop:
   ldob     (g0), g1
   addo     g1, g2, g2
   cmpinco  g0, g3, g3
   bl.t     loop
Execution:
```

Clock	REGop	MEMop	CTRLop
1		ldob	
2		:	
3		:	
4	addo		bl.t
5	cmpinco		:
6		ldob	

```
opt_loop:
   ldob     (g0), g1
   cmpinco  g0, g3, g3
   addo     g1, g2, g2
   bl.t     opt_loop
Execution:
```

Clock	REGop	MEMop	CTRLop
1		ldob	
2	cmpinco	:	
3		:	bl.t
4	addo		:
5		ldob	

Page E-41, Example E-1

The two cmpinco instructions originally read:

```
cmpinco g0, g3, g0
```

The corrected text reads:

```
cmpinco g0, g3, g3
```

Example E-3. Unrolling Loops (Checksum)

```
      -- initialize --
loop:
  ldob          (g0), g1
  addo          g1, g2, g2
  cmpinco       g0, g3, g3
  bl.t          loop
  ret
```

Execution:

Clock	REGop	MEMop	CTRLop
1		ldob	
2		:	
3		:	
4	addo		bl.t
5	cmpinco		:
6		ldob	

```
      -- initialize --
opt_loop:
  ldob          (g0), g1
  cmpinco       g0, g3, g3
  addo          g4, g2, g2
  bge.f         exit1
  ldob          (g0), g4
  cmpinco       g0, g3, g3
  addo          g1, g2, g2
  bl.t          opt_loop
  exit2:
  addo          g4, g2, g2
  ret
  exit1:
  addo          g1,g2,g2
  ret
```

Execution:

Clock	REGop	MEMop	CTRLop
1		ldob g1	
2	cmpinco	:	bge.f
3	addo g4	:	:
4		ldob g4	
5	cmpinco	:	bl.t
6	addo g1	:	:
7		ldob g1	

Page E-44, Example E-3

The three cmpinco instructions originally read:

```
cmpinco g0, g3, g0
```

The corrected text reads:

```
cmpinco g0, g3, g3
```

E.2.7.5 Enabling Constant Parallel Issue

As described in section E.2.1, “Parallel Issue” (pg. E-13), certain sequences of machine-type instructions can be executed in parallel, such as REG-MEM, REG-MEM-CTRL, MEM-CTRL. In Example E-4 the checksum loop is repeated. Another clock is eliminated by reordering code for parallel issue.

Example E-4. Order for Parallelism (Checksum)

```
      -- initialize --
loop:
  ldob          (g0), g1
  addo          g1, g2, g2
  cmpinco       g0, g3, g3
  bl.t          loop
  ret
```

Execution:

Clock	REGop	MEMop	CTRLop
1		ldob	
2		:	
3		:	
4	addo		bl.t
5	cmpinco		:
6		ldob	

```
      -- initialize --
opt_loop:
   addo         g4, g2, g2
   ldob         (g0), g1
   cmpinco      g0, g3, g3
   bge.f        exit1
   ldob         (g0), g4
   cmpinco      g0, g3, g3
   addo         g1, g2, g2
   bl.t         opt_loop
exit2:
   addo         g4, g2, g2
   ret
exit1:
   addo         g1,g2,g2
   ret
```

Execution:

Clock	REGop	MEMop	CTRLop
1	addo g4	ldob g1	bge.f
2	cmpinco	:	:
3		ldob g4	
4	cmpinco	:	bl.t
5	addo g1	:	:
6	addo g4	ldob g1	

E.2.7.6 Alternating from Side to Side

The i960 Hx processor can sustain execution of two instructions per clock. To maximize this capability, try to start instructions in two of the three pipelines each clock. To increase parallelism, move an instruction from a unit which has become a critical path to a unit with available clocks. The AGU performs shifts, additions and moves that can replace EU operations. Literal addressing

Page E-45, Example E-4

The three cmpinco instructions originally read:

```
cmpinco g0, g3, g0
```

The corrected text reads:

```
cmpinco g0, g3, g3
```