Intel386™ DX PROCESSOR SPECIFICATION UPDATE

Release Date: January, 1997
Order Number: 272874-002

The Intel386™ DX processor may contain design defects or errors known as errata. Characterized errata that may cause the Intel386 DX processor's behavior to deviate from published specifications are documented in this specification update.

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

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

or call in North America 1-800-879-4683, Europe 44-0-1793-431-155,
France 44-0-1793-421-777, Germany 44-0-1793-421-333, other countries 708-296-9333

CONTENTS

REVISION HISTORY

This document applies to the F-step of the dynamic Intel386™ DX processor. The errata and information that has changed in this technical note since the last revision are outlined below:

Rev. Date	Version	Description
12/23/96	002	**Specification Changes:** • IDIV instruction interrupt0 due to overflow
07/01/96	001	This is the new Specification Update document. It contains all identified errata published prior to this date.
07/29/92	1.00	**Specification Changes:** • Move from 16-bit Segment/System Register to 32-bit Destination • Clearing prefetch queue when transitioning into paging **Errata:** • TSS Limit Check • POPA/POPAD Instruction Malfunction • Fault During Task Switch with Single Step Enabled • Incorrect Exception by a Page Faulting LOCKed Instruction **Specification Clarifications:** • Read cycles require valid data bus levels • Use of ESP as a base register with CALL, PUSH, and POP Instructions • Use of code breaks to debug 86/286 Operating Systems • Use of ESP in 16-bit Code with 32-bit Interrupt Handlers • Debug breakpoints

PREFACE

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

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

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

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

Affected Documents/Related Documents

Title	Order
Intel386™ DX Microprocessor datasheet	241267-001

Nomenclature

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

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

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

Documentation Changes include typos, errors, or omissions from the current published specifications. These 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 Intel386 DX processor product. Intel may fix some of the errata in a future stepping of the component, and account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

Codes Used in Summary Table

Stepping

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

Page

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

Status

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

Row

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

Errata

No.	Rev. Date	Steppings			Page	Status	ERRATA
		F0					
1	7/29/92	X			9	NoFix	TSS Limit Check
2	7/29/92	X			9	NoFix	POPA/POPAD Instruction Malfunction
3	7/29/92	X			10	NoFix	Fault During Task Switch With Single Step Enabled
4	7/29/92	X			11	NoFix	Incorrect Exception By A Page Faulting Locked Instruction

Specification Changes

No.	Steppings			Page	Status	SPECIFICATION CHANGES
	F0					
1	X			12	Doc	Move From 16-Bit Segment/System Register To 32-Bit Destination
2	X			12	Doc	IDIV Instruction Interrupt0 Due To Overflow
3	X			12	Doc	Clearing Prefetch Queue When Transitioning Into Paging

Specification Clarifications

No.	Steppings			Page	Status	SPECIFICATION CLARIFICATIONS
	F0					
1	X			13	Doc	Read Cycles Require Valid Data Bus Levels
2	X			13	Doc	Use Of ESP As A Base Register With CALL, PUSH, And POP Instructions
3	X			14	Doc	Use Of Code Breaks To Debug 86/286 Operating Systems
4	X			14	Doc	Use Of ESP In 16-Bit Code With 32-Bit Interrupt Handlers
5	X			14	Doc	Debug Breakpoints
6	X			15	Doc	Breakpoint Exceptions

Documentation Changes

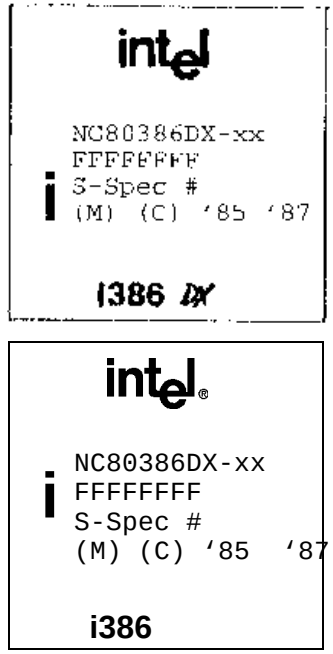
No.	Document Revision	Page	Status	DOCUMENTATION CHANGES
				None for this revision of this specification update.

IDENTIFICATION INFORMATION

Markings

Intel386 DX processors may be identified electrically according to device type and stepping. Refer to the data sheet for instructions on how to obtain the identifier number.

The F0 part is identified with the following marks:



Intel386™ DX CPU-F0 (CHMOS IV)

Legend:

MHz	S-Spec#	S-Spec#
16	S X6894	S X689
20	S X6904	S X690

Legend:

25	S X6914	S X691
25	S X6914	S X692

ERRATA

1. TSS Limit Check

PROBLEM: The TSS limit is checked to determine if the segment being set aside is large enough. If the TSS limit is not large enough, an exception #10 is generated to indicate the error. The current microcode generates the exception for limits less than 101 bytes. This exception should be generated for limits less than 103 bytes. By not generating exception #10 for TSS limits of 101 and 102 bytes, the microcode allows task state segments that do not contain the I/O Protection Bit Map Offset.

IMPLICATION: The omission of the I/O Protection Bit Map Offset leads to a fault whenever an attempt is made to access the I/O Protection Bit Map Offset. These accesses occur in systems running protected mode and attempting to perform task switches.

WORKAROUND: The problem is avoided by verifying that a task state segment is generated with a minimum limit of 103 (67H) bytes. A task state segment of this size is large enough to contain all of the required fields.

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

2. POPA/POPAD Instruction Malfunction

PROBLEM: Under certain conditions when the POPA or POPAD instruction is used, the Intel386 DX CPU executes improperly. The Intel386 DX CPU inadvertently corrupts the EAX register when either the POPA or POPAD instruction is immediately followed by an instruction that uses a base address register AND an additional register other than EAX or AX as an index register to form its effective address.

The following sample code is an example of the problem:

```
MOV EDX,4
POPAD
MOV EBX, dword ptr [EDX + EBX*4]
```

Additionally, whenever a POPA (16-bit version) instruction is immediately followed by an instruction which uses the EAX (32-bit register) as a base OR index register to form its effective address, the Intel386 DX Processor will hang.

The following sample code is an example of the problem:

```
MOV EAX,4
POPA
MOV EBX, dword ptr [EAX]
```

IMPLICATION: Depending on the above conditions, the EAX register will contain an undefined value or the processor will stop execution. Proper operation of the processor cannot be guaranteed after this sequence is executed until a hardware reset occurs. This sequence of instructions can occur in the Real, Protected and Virtual 86 modes of the Intel386 DX CPU.

WORKAROUND: Never execute the described instruction sequences. A workaround which has proven to be successful in all cases is to insert a NOP instruction after every POPA(D) instruction. Examples are shown below:

EXAMPLE 1

```
MOV EDX,4
POPAD
NOP
MOV EBX, dword ptr [EDX + EBX*4
```

EXAMPLE 2

```
MOV EAX,4
POPA
NOP
MOV EBX, dword ptr [EAX]
```

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

3. Fault During Task Switch with Single Step Enabled

PROBLEM: If a fault occurs during a task switch and if single step is enabled (TF=1) the potential exists for the new task to be entered with single step enabled. This will only happen if the following three conditions are met:

1. The TF (single step) flag is set to 1 when the task switch occurs.
2. A fault is detected while the processor is saving the state of the current task.
3. A task gate is used to handle the fault.

IMPLICATION: The problem only occurs during the debug phase of code generation. Application code is not affected since single step would not be enabled while executing

code. Also, the only faults possible during the problem window are exception 10 and exception 14. Exception 10 will only be encountered during this window if the TSS limit is altered *after* the task was entered. Exception 14 will only be encountered if the current page is marked “Not Present.” Neither of these procedures is recommended programming practice.

WORKAROUND: Eliminate one of the necessary conditions. The easiest is to follow recommended programming techniques. Do not alter the TSS limit of the current task and do not mark the current page “Not Present.”

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

4. Incorrect Exception by a Page Faulting LOCKed Instruction

PROBLEM: When a LOCKed instruction crosses a page boundary and the next page is marked “Not Present,” a page fault should be generated (exception 14), but instead the CPU may generate an invalid opcode violation (exception 6). This only occurs if the page boundary falls between any two bytes of the entire LOCKed instruction.

IMPLICATION: The only effect this problem has on the system is that it may generate the wrong exception if above conditions are met. This behavior is expected more on multi-processing and in multi-thread code where use of LOCK instruction is common.

On sensing the fault, the CPU saves the CS and EIP values before transferring control to the handler. As faults are fully restartable, the faulting program can be resumed after “fixing” the cause of the exception.

WORKAROUND: There are at least two ways to avoid this problem.

1. Rewrite the exception 6 handler to check for the following conditions:
   a. Did any part of the instruction (prefix, opcode, operand) cross a page boundary?
   b. Does the instruction have a LOCK prefix?
   c. Is the next page where the fault occurred marked not present?

   If these conditions are met, the modified exception 6 handler can invoke a page fault handler to bring in the faulting page, and simply restart the instruction by executing an IRET instruction.
2. Avoid putting a LOCKed instruction on a page boundary with the next page being not present.

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

SPECIFICATION CHANGES

1. Move from 16-bit Segment/System Register to 32-bit Destination

ISSUE: This clarifies how certain instructions (which imply a 16-bit operand size) behave with various operands and operand sizes. These instructions are: MOV r/ml6,Sreg; STR r/ml6; SLDT r/ml6; and SMSW r/ml6. When a 32-bit operand size is selected, and the destination is a register, the 16-bit source operand is copied into the lower 16 bits of the destination register, and the upper 16 bits of the destination register are undefined. With a 16-bit operand size and a register operand, only the lower 16 bits of the destination register are affected (the upper 16 bits remain unchanged). With a memory operand, the source is written to memory as a 16-bit quantity, regardless of operand size. Thus, 32-bit software should always treat the destination as 16-bits, and mask bits 16-31 if necessary.

2. IDIV Instruction Interrupt0 Due To Overflow

ISSUE: The IDIV instruction operands need to be sign-extended to avoid incurring an overflow condition (interrupt0 exception).

An extremely limited subset of IDIV calculations may not properly generate an overflow interrupt0 exception subsequent to the actual occurrence of an overflow calculation. This inhibited interrupt0 exception condition results in an incorrect IDIV quotient calculation. The IDIV instruction does not incur the overflow condition when sign-extension is conducted before the operation. Direct assembly language programming should also perform sign-extension prior to executing IDIV to preclude the occurrence of the overflow condition (sign-extension of the divisor is used to prevent the overflow condition, sign-extension of the dividend is used to satisfy the IDIV instruction format). The Intel386 processor instruction set incorporates several commands to facilitate sign-extension. The CBW, CWD(CWDE), and CDQ instructions are often utilized to automatically sign-extend byte, word, and double-word data.

Fortran and C require that arithmetic operations be performed on operands of equal length; some compilers may already either extend the shorter operand or extend both operands to a common larger size. To accommodate both positive and negative numbers, these compilers typically establish this equal length by performing sign-extension.

3. Clearing Prefetch Queue When Transitioning Into Paging

ISSUE: After setting the PG (paging enable) bit in CR0, a short, aligned jmp instruction should be executed to clear the prefetch queue. If this is not done, a queue corruption problem may occur. However, if the addresses before and after enabling paging are the same, the short, aligned jmp instruction is unnecessary.

SPECIFICATION CLARIFICATIONS

1. Read Cycles Require Valid Data Bus Levels

ISSUE: The Intel386 DX CPU requires that all data bus pins be at a valid logic state (high or low) at the end of each read cycle, when READY# is asserted. The system MUST be designed to meet this requirement. Therefore, do NOT allow any data lines to be floating when the read cycle completes. This requirement applies to the following bus cycles: Memory Data Read, Memory Code Read, I/O Data Read, and Interrupt Acknowledge (see datasheet table 5-2. Bus Cycle Definition).

If the device being read is a 32-bit device, such as a 32-bit memory, the system should present 32-bits of data to the Intel386 DX CPU even if not all of the Intel386 DX CPU byte enables are asserted.

If the device being read is a 16-bit or an 8-bit device, however, pullup resistors can be used to guarantee valid logic levels on the upper data lines, which otherwise would be floating. Note that bus cycles to 16-bit and 8-bit devices typically include several wait states, but always calculate the effects of R-C time constants to ensure the pullups will drive proper logic levels onto the bus within the time required.

2. Use of ESP as a Base Register With CALL, PUSH, and POP Instructions

ISSUE: This clarifies how ESP behaves with instructions that implicitly reference the stack and explicitly reference another location in memory using ESP as a base register.

Instruction	Explicit Memory Reference uses the ESP value	ESP value used as base
CALL-indirect thru-memory	before decrementing	old ESP
PUSH-from memory	before decrementing	old ESP
POP-to memory	after incrementing	new ESP

This is consistent in that the CALL-indirect-thru-memory and the PUSH-from-memory both use the same ESP value.

Furthermore, the relation between PUSH-from-memory and POP-to-memory is such that it allows the instruction sequence:

```
PUSH [ESP+n]
POP [ESP+n]
```

to have the desirable property of both instructions referencing the same memory location.

3. Use of Code Breaks to Debug 86/286 Operating Systems

ISSUE: The RF bit in the EFLAGS register is cleared by a 16-bit IRET, making it difficult to use the on-chip debug registers to set code breakpoints to debug 16-bit operating systems. Data breakpoints work fine in all cases, and code breakpoints work fine as long as all interrupt handlers are 32-bits and return with 32-bit IRETs or task switches. In 16-bit environments, software debuggers should use the CC (single byte INT 3 instruction) to place software breakpoints in code.

4. Use Of ESP In 16-Bit Code With 32-Bit Interrupt Handlers

ISSUE: When a 32-bit IRET is used to return to another privilege level, and the old level uses a 4G stack (D/B bit in the segment register = 1), while the new level uses a 64k stack (D/B bit = 0), then only the lower word of ESP is updated. The upper word remains unchanged. This is fine for pure 16-bit code, as well as pure 32-bit code. However, when 32-bit interrupt handlers are present, 16-bit code should avoid any dependence on the upper word of ESP. No changes are necessary in existing 16-bit code, since the only way to access ESP in USE16 segments is through the 32-bit address size prefix.

5. Debug Breakpoints

ISSUE: Under certain debug conditions, the fetches of interruptable instructions (ESC and REP + string instructions) are repeated. This problem occurs in the following cases:

Case 1:

Code breakpoint set on the instruction following an interruptable instruction

If the interruptable instruction is an ESC instruction waiting for the BUSY# signal to go inactive (indicating the Intel387 DX math co-processor has completed the previous ESC instruction), the ESC instruction will be repeatedly fetched for the duration of the wait. If the interruptable instruction is a REP + string instruction, the instruction will be fetched once for each iteration indicated by the REP. In both cases, once the interruptable instruction is completed, control will be transferred to the user's debug handler.

Case 2:

Single stepping through ESC instructions

In the case where TF=1 and the BUSY# signal (from a previous ESC instruction) is still active, the single step handler will be entered with the ESC instruction not executed and with the EIP still pointing to the same instruction. On returning from the handler, the same instruction, ESC, will be fetched again. In the event the BUSY# signal is still active after the second fetch of the ESC instruction, the debug handler will be entered once again. After completely executing the ESC instruction, the debug handler will be entered with the EIP pointing to the next instruction.

Neither case alters the result of the execution. The only implication is that users may see the additional fetches. The extra fetches of cases 1 and 2 could show up to users who monitor the address bus during debug or to users who have an ICE part and observe the trace data buffer or the ICE0-3 pins. Users who single step on an instruction mix that contains long ESC instructions could notice multiple entries to the debug handler for the same instruction.

6. Breakpoint Exceptions

ISSUE: There are four circumstances under which a breakpoint exception will be missed. All code will continue to run properly - all instructions will be executed completely and correctly.

Case 1

If a data breakpoint is set to a mem16 operand of a VERR, VERW, LSL, or LAR instruction and the segment with the selector at mem16 is not accessible, the breakpoint will be missed.

Case 2

The loop of a REP MOVS instruction consists of two bus cycles, a read cycle and a write cycle. The breakpoint exception will be missed when the data breakpoint is set to the operand of the write cycle of the REP MOVS instruction, and the read cycle of the next iteration of the REP MOVS instruction gets a fault.

Case 3

Normally a CODE breakpoint after a MOV or POP to SS instruction will cause a debug exception after the following instruction. However, if the following instruction has an execution time of more than 2 clocks, the debug exception will not occur.

Case 4

Normally a DATA breakpoint set to the operand of a MOV to SS instruction will cause a debug exception after the following instruction is executed. However if the execution time of the following instruction is more than 2 clocks, the debug exception will not occur.

DOCUMENTATION CHANGES

None for this revision of this specification update.