ERRATA FOR IDT WINCHIP 2, 2A AND 3

1. INTRODUCTION

This document describes the *IDT WinChip 2, 2A and 3* processor *errata*: differences between the actual *IDT WinChip* behavior and the expected results.

The *IDT WinChip C6* is the predecessor of the *IDT WinChip 2* and has a different set of errata described in a separate document. Use the Model ID to distinguish the *IDT WinChip 2, 2A and 3* from its predecessor. The Model ID is part of the processor identification signature as returned by the CPUID instruction or in EDX following Reset.

The *IDT WinChip 2* and *IDT WinChip 2A* have different errata (the *IDT WinChip 2A* and *IDT WinChip 3* have the same errata). Use the Stepping ID to distinguish between versions of the processor. Table C-1 describes the values returned.

Table C-1

Code	Processor	Type ID	Family ID	Model ID	Stepping ID
	IDT WinChip C6	0	5	4	Varies
B	IDT WinChip 2	0	5	8	5
C	IDT WinChip 2A	0	5	8	7, 8, or 9
D	IDT WinChip 3	0	5	9	Varies

Tables C-2 and C-3 describe the codes used to define the status and action plan of each individual erratum.

Table C-2

Status Code	***Description***
X	This version of the processor has the erratum
N/A	This erratum does not apply to this version of the processor
	Fixed in this version of processor

Table C-3

Plan Code	*Description*
Fixed	This erratum is fixed in the latest version of the processor
Fix	This erratum may be fixed in some future version of the processor
NoFix	There are no plans to fix this erratum in future steppings of the *IDT WinChip 2* processor

The *IDT WinChip 2* processor errata described here is complete and is described in considerable detail. This detailed exposure is consistent with the Centaur philosophy of openness. However, the errata do not, as far as we know, represent a real compatibility exposure. The errata were found only by specialized design-verification tests, which perform unnatural acts.

Table C-4 summarizes the errata and provides the codes which describes the status and action plan for each individual erratum. The details of each erratum are described in subsequent sections.

Table C-4

ID	*Code (see Table C-1)*			*Plans*	*Errata*
	B	C	D		
F-1	X	X	X	NoFix	FPU environment may be different from P55 in certain cases
F-2	X	X	X	NoFix	FPU transcendental may return a denormal value without underflow
F-3	X			Fixed	FPU FIST,FISTP do not handle overflow correctly
F-4	X			Fixed	FPU compare of denormal memory operand may not set flags correctly
F-5	X			Fixed	FPU FIST,FISTP of negative unnormal data may be wrong in the least significant bit
F-6	X	X	X	Fix	FPU FPREM1 of denormal data may be wrong under certain conditions
I-1	X	X	X	NoFix	INTR is missed if deasserted while processor executes consecutive STIs
I-2	X	X	X	NoFix	Certain invalid instructions may result in page

ID	Code (see Table C-1)			Plans	Errata
	B	C	D		
					faults instead of INT6
I-3	X	X	X	NoFix	BUSCHK# does not work as in P54 in certain cases
I-4	X	X	X	NoFix	Inconsistent fault when length limit exceeded with invalid LOCK prefix
I-5	X			Fixed	BIST return code (EAX) may indicate false failures
I-6	N/A	X	X	Fix	MSR FCR4[5:2] may not return correct value
D-1	X	X		Fixed	3DNOW! 24-bit precision reciprocal square root is wrong in certain rare cases
B-1	X			Fixed	Setting FCR.DTLOCK to '0' may result in hang

2. FLOATING-POINT ERRATA

F-1. FPU environment may be different from P55 in certain cases

PROBLEM: An FSTP QWORD that crosses a page boundary and gets a page fault will set the floating point environment, technically it should not. An FRSTOR that crosses a segment limit of 4GB will partially load floating-point state.

IMPLICATION: These errata are not believed to occur in any application or operating system.

WORKAROUND: None

F-2. FPU transcendental may return a denormal value without underflow

PROBLEM: The transcendental instructions (FSIN, FCOS, FSINCOS, FPTAN, FPATAN, FYL2X, FYL2XP1, F2XM1) may, in some cases, return a denormal value without underflow. This can occur whenever the internal approximation algorithm computes a precise denormal result (i.e. no loss of precision occurred in the computation of the approximation).

In these cases, the transcendental instructions behave like the arithmetic instructions. This is technically incorrect, since transcendental approximations are inherently inexact.

If the transcendental approximation algorithm computes an exact denormal result, UE is not set in FPSW, and if UE is unmasked, the unmasked underflow response does not occur.

Examples:

```
FSIN(denormal x) = x with no underflow
FYL2X(2, denormal y) = y with no underflow
```

IMPLICATION: Applications that run with the underflow exception masked, and do not check the UE bit in FPSW (virtually all applications fall into this category) are unaffected.

WORKAROUND: In applications that do respond to underflow, this condition can be detected by examining result of a transcendental operation with FXAM to see if it is denormal, and if it is, handling that case as underflow.

For applications that expect the unmasked underflow response, the translation of the value to the middle of the normal range can computed without loss of precision in user code:

```
if (denormal result)
result = result * 2 ^ 24576
```

F-3. FPU FIST,FISTP do not handle overflow correctly

PROBLEM: The FIST and FISTP instructions will not handle overflow correctly when the following conditions are met:

- the operand of FIST/FISTP is positive and
- the rounding control in the Floating Point control word is either '*up*' or '*to nearest*' and
- the rounded operand does not fit in the destination integer and
- the rounded operand would have fit in the destination integer if rounding had been in the other direction and
- the invalid-arithmetic-operand exception is masked

The processor will store the *maximum integer* value and set both FPSW.IE and FPSW.PE (and raise precision exception if it is unmasked). The correct processor response is to store the *integer indefinite* to memory and set only FPSW.IE; it should not set or raise precision exception. The following table describes the incorrect processor response based on the Floating Point control word:

	FPCW.IE masked	FPCW.IE unmasked
FPCW.PE masked	Stores *maximum integer* instead of *integer indefinite*. Sets FPSW.PE.	Correct response
FPCW.PE unmasked	Stores *maximum integer* instead of *integer indefinite*. Sets FPSW.PE and raises FPSW.PE exception.	Correct response

The following table specifies the range of operands 'x' for which the processor stores the wrong result:

	FPCW.RC='up'	FPCW.RC='to nearest'
FIST[P] m16int	2^15 – 1 < x < 2^15	2^15 – 0.5 <= x < 2^15
FIST[P] m32int	2^31 – 1 < x < 2^31	2^31 – 0.5 <= x < 2^31
FISTP m64int	2^63 – 1 < x < 2^63	2^63 – 0.5 <= x < 2^63

IMPLICATION: Applications relying on the integer indefinite being stored when FIST/FISTP overflows will fail. This erratum has not been determined to cause any problems during compatibility testing. Virtually all applications run with precision exception masked, so it is unlikely there will be compatibility problems due to an unexpected precision exception.

WORKAROUND: Any of these workarounds will avoid the problem:

- Set the rounding control in the Floating Point control word (FPCW.RC) to round *'toward zero'* or *'down'*.
- Run with invalid operation exception unmasked.
- Check the range of the operand and avoid executing FIST/FISTP when the input operand is greater than MAXINT-1.

F-4. FPU compare of denormal memory operand may not set flags correctly

PROBLEM: The C0 or C3 flag in the floating-point status word may be cleared incorrectly after a floating-point compare instruction if the operand in the stack top (ST) was a denormal operand in single or double precision previously loaded from memory. This erratum will occur when the following conditions are met:

- an FLD m32real or FLD m64real instruction loads a denormal value (single- or double-real memory operand) onto the top of stack (call this the A-operand)

- the A-operand is not operated on, except perhaps copied by FLD or FXCH or modified by FABS or FCHS
- an FCOM, FCOMP, FCOMPP, FUCOM, FUCOMP or FUCOMPP instruction compares the A-operand in the stack top with another operand (call this the B-operand) on the register stack or in memory
- the B-operand is not a NaN, not an unsupported format, and not a single or double-real denormal in memory, nor was it loaded from such a denormal (it is unlike the A-operand in this respect)
- the A-operand has the same sign as the B-operand
- the magnitude of the B-operand is greater than or equal to the A-operand but less than the smallest normal value that can be represented in the precision in which the A-operand was stored (single or double-precision)

The processor will incorrectly clear both C0 and C3 in the floating-point status word, indicating that the A-operand is greater than the B-operand. The correct processor response is to set either C0 or C3 indicating that the A-operand is less than or equal to the B-operand.

IMPLICATION: There is a small range of numbers in the single or double-real range which is affected by this erratum. It is unlikely that both operands to a floating-point compare would fall in this range. In the event that an application encounters this erratum it would cause a conditional branch to be evaluated incorrectly. This erratum has not been determined to cause any problems during compatibility testing.

WORKAROUND: Run with denormal operand exception unmasked. The exception handler could determine if the exception is due to the load of a denormal operand and, if so, add zero to the operand on the top of stack and resume execution. The FADD operation causes the processor to adjust the operand on the top of stack and avoids this erratum on a subsequent floating-point compare.

F-5. FPU FIST,FISTP of negative unnormal data may be wrong in the least significant bit

PROBLEM: The FIST and FISTP instructions do not store the correct result for certain values of unnormal data.

- an FLD m64real instruction loads an unnormal (unsupported format) value (double-real memory operand) onto the top of stack
- this value is not operated on, except perhaps copied by FLD or FXCH or modified by FABS and/or FCHS
- a FIST or FISTP stores the value to memory (m16int, m32int or m64int)
- the operand is the maximum negative integer that fits in the destination (m16int, m32int or m64int)
- due to this erratum the data stored is incorrect in the least significant bit ('1' instead of '0')

The values that produce the wrong result are very special, as summarized by the following table:

	Register value	Incorrect output
FIST[P] m16int	0xC00E0000xxxxxxxxxxxx	0x8001
FIST[P] m32int	0xC01E00000000xxxxxxxx	0x80000001
FISTP m64int	0xC03E0000000000000000	0x8000000000000001

IMPLICATION: The numbers affected by this erratum are in a small range of the unnormals (which is a subset of the unsupported formats). These numbers are never generated by the processor as the result of an arithmetic operation and are therefore unlikely to ever be the operand of a FIST or FISTP instruction. This erratum has not been determined to cause any problems during compatibility testing.

WORKAROUND: None needed, see above.

F-6. FPU FPREM1 of denormal data may be wrong under certain conditions

PROBLEM: The FPREM1 instruction will produce the wrong result under the following conditions.

- An operand of the FPREM1 instruction is a denormal or pseudodenormal value
- The other operand is normal, denormal or pseudodenormal value
- FPSW.PE is '1'
- FPCW.PE is '1', masked precision exception
- FPCW.DE is '1' masked denormalized operand exception
- The exponent difference between the operands is less than 64
- This is the final reduction (FPSW.C2 = '0')
- The quotient of ST and ST(1) is precise and exactly half-way between two integers (the fractional part of the quotient is precisely 0.5).
- due to this erratum the processor may adjust the result attempting to roundup when it shouldn't or may not roundup when it should

IMPLICATION: Denormals and pseudodenormals are rarely the input to FPREM1, making the exact conditions that may cause an incorrect result very unlikely. This erratum has not been determined to cause any problems during compatibility testing.

WORKAROUND: A denormal operand could be detected by software and corrective action taken to avoid this erratum if the denormalized operand exception is unmasked (FPCW.DE = '0').

3. INTEGER ERRATA

I-1. INTR is missed if deasserted while processor executes consecutive STIs

PROBLEM: If the processor is executing a sequence of more than one STI and an INTR is asserted and then deasserted before the processor ends the sequence of STIs then INTR will not be recognized.

IMPLICATION:

This situation should not happen on a PC system because:

- INTR is level sensitive, the interrupt controller must latch interrupts from devices and assert INTR until the processor acknowledges.
- There is no reason for software to use consecutive STIs.

WORKAROUND: None needed, see above.

I-2. Certain invalid instructions may result in page faults instead of INT6

PROBLEM: The illegal opcode 0F, 71h 72h and 73h with ModR/m memory (Group A MMX shift) is not flagged as an invalid opcode if the instruction crosses a page boundary and the second page faults, the processor presents a page fault exception instead

IMPLICATION: If this were to happen the operating system will allocate the second page and resume the faulting task which would then get the invalid opcode exception (INT6).

WORKAROUND: None needed, see above.

I-3. BUSCHK# does not work as in P54 in certain cases

PROBLEM: If BUSHCK# and SMI# are asserted while in HALT state the processor correctly takes BUSCHK# first but Auto Halt Restart slot in SMRAM is not set when SMM is entered. A similar BUSCHK# erratum is that the ESI I/O restart slot may be incorrect if BUSCHK# and SMI# are asserted during an I/O operation.

IMPLICATION: BUSCHK# is not used in PC systems.

WORKAROUND: None needed in a PC system, see above.

I-4. Inconsistent fault when length limit exceeded with invalid LOCK prefix

PROBLEM: The LOCK prefix (0xF0) is valid only for certain instructions. The LOCK prefix is not valid, for example, when the operands are all in registers. The correct processor response for such an invalid use of the LOCK prefix is to generate an Invalid Opcode exception (INT 6).

The use of redundant prefixes could result in the instruction length exceeding 15 bytes; the correct processor response is to generate a General Protection fault (INT 13).

When an instruction has an invalid LOCK prefix *and* the length exceeds 15 bytes the resulting exception (INT 6 or INT 13) is implementation-dependent. The Pentium processor (P54) responds with INT 6 whereas the Pentium Processor with MMX Technology (P55) responds with INT 13. The IDT WinChip 2 processor response is not consistent, for some opcodes it responds with INT 6 (like a P54) and for others with INT 13 (like a P55).

IMPLICATION: This erratum has not been determined to cause any problems during compatibility testing.

WORKAROUND: Do not rely on implementation-dependent behavior.

I-5. BIST return code (EAX) may indicate false failures

PROBLEM: The Built-in Self Test (BIST) can be requested as part of the *IDT WinChip 2* processor reset sequence (INIT asserted as RESET deasserted). The BIST result is indicated by a code in EAX. Normally EAX is zero after reset. A 'zero' in EAX after BIST Reset means that no failures were detected. Any value other than 'zero' indicates an error has occurred during BIST. Due to this erratum the processor may return a value other than 'zero' in EAX, even though no failures were detected.

- Bit 15 of EAX may be set to 'one', falsely indicating an error that does not exist

IMPLICATION: Certain newer motherboards in PC systems request BIST. In these systems the BIOS may report false BIST failures due to this erratum.

WORKAROUND: In systems where BIST is desired the following sequence will avoid the problem:

- RESET the processor normally (this initializes part of the machine which needs to happen to avoid this erratum).
- Check a location in volatile RAM to see if BIST has been requested (use a special signature value that is not expected in volatile RAM when the system powers up)
- If the location in volatile RAM does not have the special signature, then store the special value and request BIST
- Otherwise (special signature found in volatile RAM) then EAX has the BIST return code.

I-6. MSR FCR4[5:2] may not return correct value

PROBLEM: In the *IDT WinChip 2A* and *IDT WinChip 3* the read-only machine specific register FCR4 contains two fields which provide the bus frequency multiplier and divider as selected by pins BF2, BF1, and BF0. The *IDT WinChip 2* does not support fractional bus ratios, has a different FCR4 and is not affected by this erratum. FCR4[5:2] should return the bus frequency divider but due to this erratum the value returned in this field may not be correct.

IMPLICATION: System software that relies on FCR4[5:2] to determine the bus frequency ratio may get the wrong result.

WORKAROUND: System software should first check to see if the processor is an *IDT WinChip 2A* or *IDT WinChip 3* by using the CPUID instruction. For these processors the bus frequency divider should be read from MSR 0x0147 bits 26:23.

4. 3DNOW! ERRATA

D-1. 3DNOW! 24-bit precision reciprocal square root is wrong in certain rare cases

PROBLEM: The PFRSQRT instruction returns an estimate of the reciprocal square root of the input number. If software requires a more precise answer it can refine the answer by using the PFRSQIT1 and PFRCPIT2 instructions using the original number and the square of the estimate provided by PFRSQRT (this computes a Newton-Raphson iteration). Due to this erratum there are certain cases of the input number for which the combination of PFRSQRT and PFRSQIT1 instructions will yield the wrong result (even though the estimate by PFRSQRT is correct). The affected input numbers are normal single precision reals (positive or negative) which have an odd exponent (biased) less than +252 and a mantissa with all '1's except for the two least significant bits, which may be '0' or '1'.

Due to this erratum the result of the Newton-Raphson iteration is one half of the correct result when the input number is one of these special cases. The numbers affected are slightly less than (within 4 ULPs) certain powers of two (… 0.125, 0.5, 2.0, 8.0, …), as follows:

Sign	Biased Exp	Mantissa
x	xxxxxxx1 (not 11111101)	11111111111111111111xx

'x' can be '0' or '1'

IMPLICATION: Most programs that use 3DNow! instructions are games, where the reciprocal square root is used in lighting calculations. For most programs this erratum is therefore insignificant since the occasional pixel with the wrong intensity will go unnoticed.

WORKAROUND: The action necessary to avoid this erratum depends on the precision required for the reciprocal square root. If 15 bit precision is sufficient, then the result of PFRSQRT is good enough. If a program requires a 24-bit precision reciprocal square root then it should use the floating point unit to compute the Newton-Raphson iteration (PFRSQRT can still be used to provide the estimate).

5. BUS ERRATA

B-1. Setting FCR.DTLOCK to '0' may result in hang

PROBLEM: The *IDT WinChip 2* processor has a mode (the default) to improve performance that allows the processor to update the *accessed* and *dirty* bits in PDE/PTE entries without locking the bus or flushing data from the D-Cache. This mode is controlled by FCR.DTLOCK, '1' by default. When FCR.DTLOCK is '0' updates to the accessed and dirty bits in PDE/PTE entries are performed using the locked read-modify-write semantics which flushes the data from the D-Cache (like Pentium processor). Due to this erratum the processor may hang when the following conditions are met:

- paging is enabled
- the *accessed* bit in a PDE/PTE entry needs to be set
- FCR.DTLOCK has been set to '0'
- there is a branch instruction in the processor pipeline

Due to this erratum the processor executes the branch one clock too soon, causing erratic behavior (most likely a hang).

IMPLICATION: This erratum should not occur in a PC system because there is no reason for system software to modify the default setting of FCR.DTLOCK.

WORKAROUND: To avoid the erratum leave FCR.DTLOCK set to '1', the default value.