B0-step 80387 Errata Description

March 11, 1987

The B0-step 80387 has five minor functional differences from the data sheet. These differences are normally not seen in applications or operating systems. They will be changed in later steppings of the 80387. This version of the 80387 is capable of running all programs written for Xenix-286, PCDOS, MSDOS, Unix-386, and RMX-286/386. No programs run on these operating systems to date have failed when run on the B0-step 80387 except for the IBM PC-AT diagnostic program which fails the 80387 due to items 4 and 6 described here and a difference between the 80387 and 8087 due to the new IEEE arithmetic standard.

The B0-step 80387 can be identified by its marking of A80387B. The earlier A1-step 80387 was marked A80387.

1. Documentation update

Early versions of the 80387 documentation may incorrectly state that bit 12 of the control word is always read as one. This is true for the A1-step but not true for the B0-step and future versions of the 80387. The correct definition is that bit 12 is cleared after **FINIT**, **FNINIT**, **FSAVE**, and hardware reset. This bit can be changed by the **FLDCW**, **FRSTOR**, and **FLDENV** instructions. The state of bit 12 in the control word has no effect on the operation of the 80387.

2. Improper tag bits and condition codes are set for psuedo-zero values

When the operand for **FXAM** is a psuedo-zero, the B0-step 80387 will deliver a wrong result. If a psuedo-zero is in a non-empty register, **FSTENV** and **FSAVE** will store a tag word with the wrong tag value for the register. This difference is not significant to most programs since the 80387 normally never encounters a psuedo-zero. The program must specifically load a psuedo-zero to encounter this problem. Psuedo-zeroes have no useful function and were therefore dropped in the final IEEE arithmetic standard and therefore are not supported by the 80387. Psuedo-zero values are supported by the 8087 and 80287.

Psuedo-zeroes exist only in extended real format. They have a non-zero exponent with a fraction of all zeroes. Use of a psuedo-zero as an operand for any 80387 arithmetic operation will cause the invalid exception. The 80387 *never* generates a psuedo-zero value. A psuedo-zero can only be entered by either an **FLD temp-real** or **FRSTOR** instruction with a psuedo-zero operand.

The **FXAM** instruction on the B0-step 80387 with a psuedo-zero operand sets the condition code bits in the status word as if the operand were a zero. The correct respond would be the unsupported encoding with C3,C2, and C0 all zero. The **FSTENV** or **FSAVE** instructions on the B0-step 80387 set the tag field in the tag word for a non-empty register containing a psuedo-zero with the incorrect value of "00", as for a valid operand, and not the correct value of "10", as for an unsupported format.

Suggested workaround

No programs are known to be affected by this errata. This errata might affect an exception handler for the invalid exception or a generalized program to print out a value. Such programs might use **FXAM** or the tag bits to see what is in the stack top. The exception handler might fail because the B0-step 80387 would indicate the psuedo-zero is a zero. To work around this problem, add code after **FXAM** to check a value marked as zero to see if it is a psuedo-zero. If the exception handler uses the tag word, add code to specifically check a register marked as valid to see if it is a psuedo-zero. These workarounds will work on A1-step 80387 and future versions of the 80387.

3. Losing an instruction with asynchronous use of FNINIT

If the B0-step 80387 is run with different clock signals for 386CLK2 and 387CLK2, and FNINIT is executed near the end of an 80387 instruction, and another 80387 instruction is executed within 17 80387 clocks of the FNINIT, the second 80387 instruction may be lost. This is normally not a problem because FNINIT is not normally executed in the middle of another instruction. If the 80387 uses the same signal for 386CLK2 and 387CLK2 this is not a problem.

FNINIT may abort the execution of any 80387 instruction in progress at any time. FNINIT does not check the BUSY# or ERROR# signals. If the FNINIT instruction is executed towards the end of the execution of the current B0-step 80387 instruction in progress, the BUSY# output will not go active. If a subsequent 80387 instruction is executed while the B0-step 80387 is still processing the FNINIT instruction, the second instruction is lost. The 80387 requires about 30 internal clocks, which is 60 periods of 387CLK2, to process FNINIT.

No programs are known to be affected by this errata. A program will normally never execute FNINIT because it destroys the state of the 80387 without checking for pending errors. Normally the FINIT instruction is used to test for a previous error before initializing the 80387. The FINIT instruction is safe because it is preceeded by the WAIT instruction which guarantees the 80387 is idle. Application programs may execute FNINIT at their beginning. This is no problem because the 80387 is already idle from the the FRSTOR instruction. If FNINIT is executed immediately after hardware RESET, this is no problem because the 80387 is already idle by the the time the 80386 is able to execute the FNINIT instruction.

Suggested workaround

If for some reason an operating system must execute FNINIT to get the 80387 ready at the expense of destroying its previous state, we recommend using two FNINIT instructions. This workaround will work on the A1-step 80387 and future versions of the 80387.

4. No invalid exception for FST ST(0) or FSTP ST(0) when ST is empty.

When executing FST ST(0) or FSTP ST(0) when the stack top is empty, the B0-step 80387 will not signal the invalid exception. This is normally not a problem because using these instructions when the stack top is empty is a programming error. If the program works, this errata situation will not happen.

One program is affected by this errata. The PC-AT 80287 diagnostic program detects this errata. The program needs to be changed anyway to allow for the change in the IEEE handling of underflow. This errata could affect program development on the B0-step 80387 because the B0-step 80387 will not correctly flag an attempt to empty an already empty stack top with FSTP ST(0) or recognize an empty register with FST ST(0). The stack is left as expected, but the programmer thought there was something in the stack top when actually it was empty. This incorrect assumption of the stack state is not signalled on the B0-step 80387.

This errata could prevent an exception handler that extends the 80387 register stack to memory from correctly working. No application program or operating system currently use this feature of the 8087, 80287, or 80387. The invalid exception handler would not be invoked when using the B0-step 80387 if the program executed FST(P) ST(0) while the stack top is empty. The exception is required to cause the invalid exception handler to load a register or several registers from the memory stack into the 80387 stack and increment the memory stack pointer.

Suggested workarounds

If the user wishes to develop an application on the B0-step 80387 with registers extended to memory, the following sequences of macro-instructions should be used instead of FST ST(0):

```
FSTP    tbyte ptr MEM
FLD     tbyte ptr MEM
```

Instead of FSTP ST(0) simply use:

```
FSTP    tbyte ptr MEM
```

These workarounds will work on the A1-step 80387 and future versions of the 80387.

5. FCOMP and FCOMPP fail to pop the stack with denormal operand and unmasked denormal error

If either operand of the FCOMP or FCOMPP instruction is a denormal and the denormal error is unmasked, the B0-step 80387 will not pop the operand like the 8087/287. If the denormal error is masked, then the pop operation is correctly performed. The FUCOMP and FUCOMPP instructions also have the same errata. Later versions of the 80387 will change the FUCOMP and FUCOMPP instructions along with the FCOMP and FCOMPP instructions.

No programs are known to be affected by this errata. This errata requires an unmasked denormal exception and a denormal operand to arise. This errata could be significant if an 8087/287 program uses an unmasked denormal exception to automatically normalize operands. The exception handler would expect the pop to occur. Since the POP did not occur the program will have one or two extra operands left on the stack. This may cause an invalid exception or NAN to occur or be generated later when the register stack overflows.

Suggested workarounds

The 80387 will automatically normalize denormal operands. The denormal exception could be masked for these programs when they are run on the 80387. If the denormal exception must remain unmasked to handle them specially, then the FCOMP and FCOMPP instructions need to be changed to a FCOM instruction followed by separate FSTP instructions.

6. FLD with a denormal operand and unmasked denormal error signals ERROR# too late

If FLD instruction with a single or double precision operand encounters a denormal and the denormal error is unmasked, the B0-step 80387 ERROR# output goes active in the same 386CLK2 period as BUSY#. Normally the ERROR# output goes active at least six 386CLK2 periods before BUSY# goes inactive. This is not a problem if the 80387 ERROR# output is directly connected to the 80386.

PC-AT compatible designs that use external logic to route the 80387 error signal through an external interrupt controller may be affected by this errata. External logic is required to extend the BUSY# output when ERROR# is active. If that logic latches the BUSY# output with the rising edge of ERROR#, the busy signal may not be extended because of this errata. The next floating-point instruction may be executed before the ERROR# signal has a chance to interrupt the 80386. The result is that the next 80387 instruction uses an incorrect machine state. The internal error will still remain active, so the second instruction after the FLD will react to the error. The error and data pointers will point at the 80387 instruction following the FLD instruction. The effect of this errata is that denormal errors might be reported at the B0-step 80387 instruction following the FLD instruction. If an exception handler is used, the result is unpredictable. If the interrupt causes the program to be stopped, then the address of the failing instruction will be wrong.

The only program known to be affected by this errata is the PC-AT diagnostic. No other PC programs tested to date use the denormal error.

Suggested workaround

Only designs that sample BUSY# with the falling edge of ERROR# are affected. If this problem must be fixed, use ERROR# active to extend BUSY#. The ERROR# output must be inhibited after the 80386 responds to the interrupt. A flip-flop can be set by the F0 output port to inhibit the extension of BUSY# by ERROR#. The next activation of BUSY# can then clear the flip-flop that inhibits the extension of BUSY# by ERROR#.