Historical Considerations

Computer arithmetic has the constraint of limited precision. This limit may determine overflows or underflows, which may result in exceptions or interrupts. Floating-point arithmetic is an approximation of real numbers, and care needs to be taken to ensure that the computer number selected is the representation closest to the actual number.

Over the years there were several disputes and problems related to computer arithmetic. Some of the most known problems are described next.

The Pentium FDIV Bug

The first incident related to the *Pentium* processors occurred in the fall of 1994, a few months after *Intel* had introduced its *Pentium* microprocessor. This started with an e-mail message from Thomas Nicely, a mathematician at *Lynchburg College* in Virginia, who was doing computations related to the distribution of prime numbers. Nicely pointed out that the chip gave incorrect answers to certain floating-point division calculations. Other users confirmed the problem and identified additional examples in which an error occurs.

Typically, the flawed chip generated a slightly inaccurate answer for only a few pairs of numbers. *Intel* claimed that only one in nine billion division operations would exhibit reduced precision, and for an average user, the error only occurs once in 27,000 years. Therefore, the probability of randomly coming across the affected numbers was very small. But computations don't necessarily involve random selections of numbers, and such errors can matter in scientific and engineering applications and even in some spreadsheet calculations. IBM Research Division sustained that spreadsheet programs can produce errors once at each 24 days.

The fault itself had occurred because of the omission of five entries in a table of 1,066 values required by the radix-4 SRT division algorithm used by the *Pentium* processor. The advantage of this algorithm is that in each step 2 bits of the quotient are obtained, speeding up the rate at which division can be performed. The five cells should have contained the constant +2, but the processor fetched a zero.

Intel corrected the problem and replaced the flawed chips, but the cost of this error was about 300 million dollars.

The Pentium II Math Bug

Just two days before the announcement of the Pentium II processor, Intel was hit by reports of a math bug in its *Pentium Pro* and the soon to be announced *Pentium* II processors. The bug was known as the *Dan-0411* bug by the news media. “Dan” is the discoverer of the bug, and 04-11 (1997) is the date when the bug was first reported. *Intel* named this bug the *Flag Erratum*.

The bug relates to operations that convert floating-point numbers into integer numbers. All floating-point numbers are stored inside of the microprocessor in an 80-bit format. When a number is loaded into the microprocessor, it is converted to an 80-bit format. Integer numbers are stored externally in two different sizes. A short integer is stored in 16 bits, and a long integer is stored in 32 bits. It is often desirable to store the floating-point numbers as integer numbers. On occasion, the converted numbers won't fit into the smaller integer format. This is when the bug occurs.

The host software is supposed to be warned by the microprocessor when such a floating-point conversion error occurs; a specific error flag is supposed to be set in a floating-point status register. If the microprocessor fails to set this flag, it does not comply with the IEEE *Floating Point Standard*. For the *Dan-0411* bug, the *Pentium* II and *Pentium Pro* fail to set this error flag in many cases.

The case of the *Ariane* rocket is a sensational example of the drastic consequences of an unhandled float-to-integer overflow. *Pentium Pro* and *Pentium* II users, on the other hand, are most likely to see the results of this bug in their graphics displays or in heavy-duty numerical analysis programs. *Intel* says ordinary users might see a temporary screen glitch on some games when this bug occurs.

The *Dan-0411* bug occurs when a large negative floating-point number is stored to memory in an integer format. Under normal operation, the largest negative integer (MAXNEG) is stored in memory when a floating-point number is too large to fit in the integer format. The FPU *Status Word* (FSW) is supposed to indicate that an *Invalid Operand Exception* occurred (FSW.IE = 1). Instead of setting this bit, the *Pentium Pro* only sets the *Precision Exception* (FSW.PE) bit. The precision-exception flag indicates that a computation can't be precisely represented by the floating-point operation – in this case, the float-to-integer store operation. In most cases, this bit is ignored by programmers. Therefore, when the conditions are met for the *Dan-0411* bug to occur, programmers may never know that an error occurred.

The *Dan-0411* bug can occur when exceptions are either masked or unmasked. In the case of masked exceptions, the correct value is stored to memory; only the FSW is incorrectly set. For unmasked exceptions, the errant behavior is more serious:

- No exception occurs. The floating-point exception handler is not invoked. Therefore, the errant condition is undetectable.
- MAXNEG is returned to memory. Storing MAXNEG to memory is an errant condition. When exceptions are unmasked, nothing is supposed to he stored to memory. This means the microprocessor is erroneously storing data to memory when no data is expected.
- In the case of the `FISTP` instruction, the floating-point value is popped from the floating-point stack. When exceptions are unmasked, the floating-point stack is supposed to remain unchanged to allow for error recovery. In this case, the value is popped from the stack and gone forever. Even if the errant condition was detectable, it would be unrecoverable after the `FISTP` instruction.

The Failure of the Ariane 5 Rocket

The French rocket *Ariane 5* was launched in Kourou, French Guyana, on 4th of June 1996. The flight of the vehicle was nominal until approximately 37 seconds after lift-off. Shortly after that time, the vehicle suddenly deviated its flight path, broke up, and exploded. A preliminary investigation of flight data showed:

- Nominal behavior of the launcher up to 36 seconds after lift-off;
- Failure of the back-up inertial reference system, followed immediately by failure of the active inertial reference system;
- Pivoting into the extreme position of the nozzles of the two solid boosters and, slightly later, of the *Vulcain* engine, causing the launcher to deviate abruptly; and
- Self-destruction of the launcher, correctly triggered by rupture of the links between the solid boosters and the core stage.

The origin of the failure was thus rapidly narrowed down to the flight control system, and more particularly to the two inertial reference systems (SRIs), which ceased to function almost simultaneously at about 36.7 seconds.

The flight control system of *Ariane 5* is a standard design. The attitude of the launcher and its movements in space are measured by an SRI. It has its own internal computer, in which angles and velocities are calculated on the basis of information from an inertial platform, with laser gyros and accelerometers. The data from the SRI are transmitted through the data bus to the on-board computer

(OBC), which executes the flight program and controls the nozzles of the solid boosters and the *Vulcain* engine, via servovalves and hydraulic actuators.

For improved reliability, there is considerable redundancy at the equipment level. Two SRIs operate in parallel, with identical hardware and software. One SRI is active, and one is in "hot" stand-by; if the OBC detects that the active SRI has failed, it immediately switches to the other one, provided that this unit is functioning properly. Likewise, there are two OBCs, and a number of other units in the flight control system are duplicated as well.

The design of the SRI used in *Ariane 5* is almost identical to that of *Ariane 4*, particularly with regard to the software. Based on the extensive documentation, the following chain of events was established, starting with the destruction of the launcher and tracing back in time toward the primary cause:

The launcher began to disintegrate at about 39 seconds because of high aerodynamic loads resulting from an angle of attack of more than 20 degrees, which led to separation of the boosters from the main stage, which in turn triggered the self-destruct system of the launcher. This angle of attack was caused by full nozzle deflections of the solid boosters and the main *Vulcain* engine. The nozzle deflections were commanded by the OBC software on the basis of data transmitted by the active SRI (SRI 2). Part of the data for that time did not consist of proper flight data, but rather showed a diagnostic bit pattern of the computer of SRI 2, which was interpreted as flight data. SRI 2 did not send correct attitude data because the unit had declared a failure due to a software exception. The OBC could not switch to the back-up SRI (SRI 1) because that unit had already ceased to function for the same reason as the SRI 2.

The internal SRI software exception was caused during execution of a data conversion from a 64-bit floating-point number to a 16-bit signed integer value. The value of the floating-point number was greater than what could be represented by a 16-bit signed integer. The result was an operand error. The data conversion instructions (in *Ada* code) were not protected from causing operand errors, although other conversions of comparable variables in the same place in the code were protected.

The operand error occurred because of an unexpected high value of an internal alignment function result, called BH (horizontal bias), which is related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time. The value of BH was much higher than expected because the early part of the trajectory of *Ariane 5* differs from that of *Ariane 4* and results in considerably higher horizontal velocity values.

Not all the conversions were protected because a maximum workload target of 80% had been set for the SRI computer. To determine the vulnerability of unprotected code, an analysis was performed on every operation that could generate an exception, including an operand error. In particular, the conversion of floating-point values to integers was analyzed; operations involving seven variables were at risk of leading to operand errors. This led to protection being added to four of the variables. The three remaining variables, including the one denoting horizontal bias, were unprotected because further reasoning indicated either that they were physically limited or that there was a large margin of safety – reasoning that in the case of the variable BH turned out to be faulty.

Although the source of the operand error has been identified, this in itself did not cause the mission to fail. The specification of the exception-handling mechanism also contributed to the failure. In the event of any kind of exception, according to the system specification, the failure should be indicated on the data bus, the failure context should be stored in an EEPROM memory, and the SRI processor should be shut down.

It was the decision to cease the processor operation that finally proved fatal. The reason behind this drastic action lies in the custom within the *Ariane* program of addressing only random hardware failures. From this point of view, exception- or error-handling mechanisms are designed for random hardware failures, which can be handled by a backup system.

Although the failure resulted from a systematic software design error, mechanisms can be introduced to relieve this type of problem. For example, the computers within the SRIs could have con-

tinued to provide their best estimates of the required attitude information. There is reason for concern that a software exception should be allowed, or even required, to cause a processor to halt while handling mission-critical equipment. Indeed, the loss of a proper software function is hazardous because the same software runs in both SRI units. In the case of *Ariane 5*, this resulted in the switching off of two still healthy critical units of equipment.