



Wrocław  
University  
of Science  
and Technology

# Can a Computer be Wrong?

## Information Technologies

Wojciech Myszka

Department of Mechanics, Materials and Biomedical Engineering

January 2023



HR EXCELLENCE IN RESEARCH



- ① Data
- ② Human (operator)
- ③ Hardware (inevitable)
- ④ Hardware
- ⑤ Manufacturer fault
- ⑥ Software

7 Bug free software

8 List of software bugs





# Data



# Measurement (Absolute error)

- ▶ I was talking about this in one of the **previous lectures**.
- ▶ One of the main sources of errors are **data**.
- ▶ Knowing the range of each value allows us to predict values of the result.
  - ▶ This can be difficult and tricky.
- ▶ However, in most cases we **assume** that the data values are **correct** and believe in computer calculations.



# Human (operator)



# Operator

- ▶ In general, it is difficult to take into account **human errors**.
- ▶ These include:
  - ▶ not understanding the problem solved by the program,
  - ▶ errors in preparing input data,
  - ▶ wrong answer to the computer prompt,
  - ▶ ...



# Hardware (inevitable)





# Number of bits

This is a quite different source of error.

1. Most of today's computers have

- ▶ 32 or 64 bits processors

2. What does this mean?



# Number of bits — range I

## 1. The biggest integer value

- ▶ 32 bits: from  $-2^{31}$  to  $2^{31} - 1$  (−2,147,483,648 to 2,147,483,647) or **two billion, one hundred forty-seven million, four hundred eighty-three thousand, six hundred forty-seven**
- ▶ 64 bits: from  $-2^{63}$  to  $2^{63} - 1$  (−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807) **nine quintillion two hundred twenty three quadrillion three hundred seventy two trillion thirty six billion eight hundred fifty four million seven hundred seventy five thousand eight hundred and seven**

## 2. What happens if our value exceeds this limits?



# Number of bits — range II

2.1 Let us assume that we are using 8 bit arithmetics (this will be simpler!)

bit no.	7	6	5	4	3	2	1	0
127	0	1	1	1	1	1	1	1
+1	0	0	0	0	0	0	0	1
-128	1	0	0	0	0	0	0	0

## Number of bits — range III

in decimal this means  $127 + 1$

and the result is  $-128!$

All integer computations are performed using so called **modulo arithmetics**

- ▶ modulo  $2^{32}$  (32 bit computer)
- ▶ modulo  $2^{64}$  (64 bit computer)
- ▶ modulo  $2^8$  (8 bit computer)

i.e., the computer takes  $n$  least significant bits from the result  
( $n = 8, 16, 32, 64, \dots$ )



# Number of bits — range (floating point numbers)

Shortly (more about this in the lecture **number 5**)

1. 32 bits computer from  $1.17549 \times 10^{-38}$  to  $3.4028235 \times 10^{38}$
2. 64 bits computer from  $2^{-1022} \approx 210^{-308}$  to approximately  $2^{1024} \approx 210^{308}$  ( $2.22507 \times 10^{-308}$  to  $1.79769 \times 10^{308}$ ).
3. If we exceed the range we will get *infinity* on one side, and zero on the other.



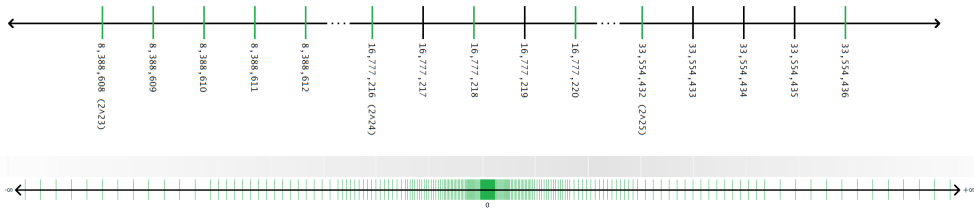
# Floating points numbers — precision

## Shortly

1. 32 bits computer: 24 significant binary digits (approx. 7.2 decimal digits)
2. 64 bits computer: 53 significant binary digits (approx. 15.9 decimal digits)
3. Beware adding numbers differing in range!
4. Remember about decimal to binary conversions errors



# Floating point numbers vs Real



0.1 or 10 cents

$$0.1_{10} = 0.0(0011) \approx 0,0001100110011001100110011 \dots$$

Taking into account only 24 significant binary digits and converting back to decimal, we obtain 0.10000000149011611938

It is quite good, but not exact.



# Error or design principle?

If you know this — it is a **design principle!**

If not — can be treated as **an error!**

In general

These are not errors!





# Hardware



# Random I

1. Computer is so complicated that if something goes wrong (e.g., overheating the processor, memory errors,...) it stops working.
2. Accidental change of data value in memory (random bit flipping caused by radiation) is very rare:

2.1 memory without ECC — approximately once per 7 years

2.2 memory with ECC — approximately once per 700 years

This changes over time, and now is less frequent than in early 70's

3. Errors during data transmission
4. Errors on disk



# Random II

That can happen, but...

There are special algorithms for detection, correction, and recovery:

- ▶ control sum,
- ▶ retransmission,
- ▶ ...



# Manufacturer fault



# Beginnings of 80x86

1. In 1978 16-bit processors were introduced

- ▶ 8086
- ▶ 80186
- ▶ 80286

2. In 1985 32-bit processors were introduced

- ▶ 80386
- ▶ 80486
- ▶ 80586 (??) name change: Pentium or P5 (1993)



# fdiv bug I

1. Story **described** by **Thomas R. Nicely**
  - ▶ math professor
  - ▶ working in computational number theory
2. **Intel** testers realized (May 1994), that new processor incorrectly computes result of one mathematical operation. They do nothing.
3. Nicely has discovered (June 1994) strange bug in newly bought Pentium processor

insted

$$\frac{4195835.0}{3145727.0} = 1.333\,820\,449\,136\,241\,002$$



# fdiv bug II

computes

$$\frac{4195835.0}{3145727.0} = 1.333\ 739\ 068\ 902\ 037\ 589$$

4. Nicely performs tests (July–November'94) and definitively confirms that bug is caused by a processor.
5. Nicely sent an email describing the error he had discovered in the Pentium floating point unit to various contacts.
6. This flaw in the Pentium FPU was quickly verified by other people around the Internet, and became known as the Pentium FDIV bug.
7. The story first appeared in the press on November 7, 1994.



## fddiv bug III

8. Publicly, Intel acknowledged the floating-point flaw, but claimed that it was not serious and would not affect most users.

Failure category and system component	Hard or Soft	FIT rate (per $10^9$ device hours)	MTBF (1 in x years)	Rate of significant failure seen by user
16 4-Mbit DRAM parts in a 60Mhz Pentium TM processor system without ECC	Soft	16	7 years	Depends upon where defect occurs and how propagated
Particle defects in PentiumTM processor	Hard	400-500	200-250 years	Depends upon where defect occurs and how propagated
16 4-Mbit DRAM parts in a 60Mhz Pentium TM processor system with ECC	Soft	160	700 years	Depends upon where defect occurs and how propagated
PC user on spreadsheet running 1,000 independent divides a day on the PentiumTM processor a	Hard	3.3	27,000 years	Less frequent than 1 in 27,000 years. Depends upon the way inaccurate result gets used





# fdiv bug IV

Class	Applications	MTBF	Impact of failure in div/rem/tran
Word processing	Microsoft Word, Wordperfect, etc.	Never	None
Spreadsheets (basic user)	123, Excel, QuattroPro (basic user runs fewer than 1000 div/day)	27,000 years	Unnoticeable
Publishing, Graphics	Print Shop, Adobe Acrobat viewers	270 years	Impact only on Viewing
Personal Money Management	Quicken, Money, Managing Your Money, Simply Money, TurboTax (fewer than 14,000 divides per day)	2,000 years	Unnoticeable
Games	X-Wing, Falcon (flight simulator), Strategy Games	270 years	Impact is benign, (since game)



# fdiv bug V

Usage	Examples	Division intensive	Impact
Standard spreadsheet analysis	Corporate finance, budget or marketing analysis,	No	None
Basic financial calculations	Present value, yield to maturity	Some	Significant only in the extreme circumstance of > 10 million divisions per day
Complex mathematical models	Black-Scholes model, Binomial model	Some	Could be significant on continuous use
Path based models and simulations	Monte Carlo risk analysis, non recombining paths	Yes	Significant unless there is a low P2 factor.



# Meltdown and Spectre



## Meltdown

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. This applies both to personal computers as well as cloud infrastructure. Luckily, there are [software patches against Meltdown](#).


 [Meltdown Paper](#)



## Spectre

Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre

Spectre is harder to exploit than Meltdown, but it is also harder to mitigate. [However, it is possible to prevent specific known exploits based on Spectre through software patches.](#)

 [Spectre Paper](#)



# In general

Computers Do Not Err



# Software



# Important note

Please do remember **the software always has errors.**



# EULA

EULA

End User License Agreement



## Example I

LIMITED WARRANTY. Except with respect to the Redistributables, which are provided “as is,” without warranty of any kind, *Company* warrants that (a) the SOFTWARE PRODUCT will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt, and (b) any Support Services provided by *Company* shall be substantially as described in applicable written materials provided to you by *Company*, and *Company* support engineers will make **commercially reasonable** efforts to solve any problem. To the extent allowed by applicable law, implied warranties on the SOFTWARE PRODUCT, if any, are limited to ninety (90) days. [...]





## Example II

NO OTHER WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, *COMPANY* AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NONINFRINGEMENT, WITH REGARD TO THE SOFTWARE PRODUCT, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. [...]



## Example III

LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL *COMPANY* OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) **ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF *COMPANY* HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, *COMPANY'S* ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR U.S.\$5.00; [...]**



# The total Cost of Poor Software Quality

According to the recent report “**The Cost of Poor Software Quality in the US: A 2020 Report**” the total Cost of Poor Software Quality (CPSQ) in the US is \$2.08 trillion (T).



# TeX I

1. T<sub>E</sub>X is a typesetting system (or a “formatting system”) which was designed and mostly written by Donald Knuth and released in 1978.
2. On Knuth’s home page one can find such excerpt:  
*This is the year when I promised to do the **seven-year cleanup** of **TeX and METAFONT** (last updated in 2014).*

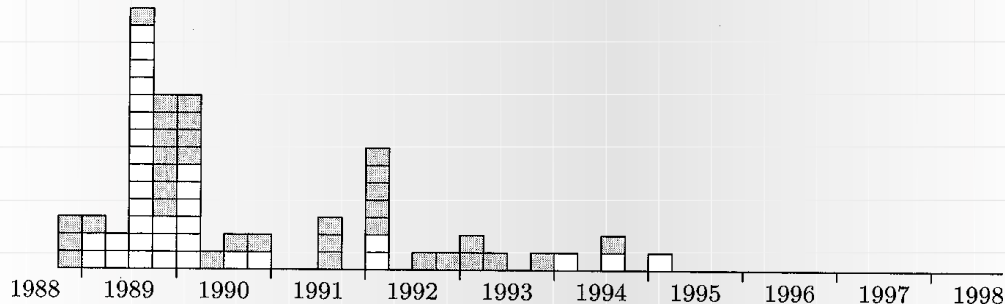


FIGURE 2. When the changes were made.



# Bug free software



# Can we build a bug free software?

The analogy between algorithms and recipes fails when it comes to issues of correctness. When a cooking or baking endeavor does not succeed there can be two reasons: 1. the “hardware” is to blame, or 2. the recipe is imprecise and unclear.

How to improve?

- ▶ testing and debugging...



# Partial and Total Correctness I

Finding an algorithmic solution consists of two tasks:

1. a specification of the set of legal inputs; and
2. the relationship between the inputs and the desired outputs.

This “relationship” is a description of an algorithm, and is used to create the algorithm.

To facilitate precise treatment of the correctness problem for algorithms, researchers distinguish between two kinds of correctness, depending upon whether termination is or is not included.





# Partial and Total Correctness II

## Partial correctness

it is said that an algorithm  $A$  is partially correct (with respect to its definition of legal inputs and desired relationship with outputs) if, for every legal input  $X$ , if  $A$  terminates when run on  $X$  then the specified relationship holds between  $X$  and the resulting output set.

Thus, a partially correct sorting algorithm might not terminate on all legal lists, but whenever it does, a correctly sorted list is the result.



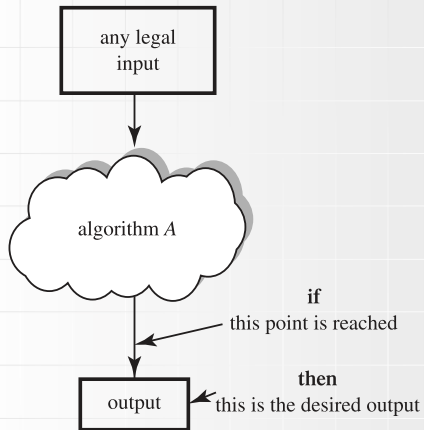
# Partial and Total Correctness III

## Total correctness

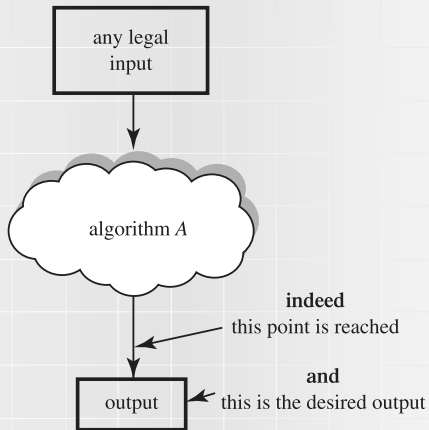
We say that  $A$  **terminates** if it halts when run on any one of the legal inputs. Both these notions taken together—partial correctness and termination—yield a totally correct algorithm, which correctly solves the algorithmic problem for every legal input: the process of running  $A$  on any such input  $X$  indeed terminates and produces outputs satisfying the desired relationship



# Partial and Total Correctness (cont.)



Partial correctness



Total correctness



# Proofing the correctness of an algorithm

1. Is very difficult
2. For every **correct** algorithm it **can** be proofed that it is correct.
3. How to do this is another matter.



# List of software bugs



# List of software bugs on Wikipedia

[https://en.wikipedia.org/wiki/List\\_of\\_software\\_bugs](https://en.wikipedia.org/wiki/List_of_software_bugs)



# Some examples I

- ▶ In the early 1960s one of the American spaceships in the Mariner series sent to Venus was lost forever at a cost of millions of dollars, due to a mistake in a flight control computer program.
- ▶ In 1981 one of the television stations covering provincial elections in Quebec, Canada, was led by its erroneous computer programs into believing that a small party, originally thought to have no chance at all, was actually leading. This information, and the consequent responses of commentators, were passed on to millions of viewers.
- ▶ In a series of incidents between 1985 and 1987, several patients received massive radiation overdoses from Therac-25 radiation-therapy systems; three of them died from resulting complications. The hardware safety interlocks from previous models had been replaced by software safety checks, but all these incidents involved programming mistakes.



## Some examples II

- ▶ Some years ago, a Danish lady received, around her 107th birthday, a computerized letter from the local school authorities with instructions as to the registration procedure for first grade in elementary school. It turned out that only two digits were allotted for the “age” field in the database.
- ▶ At the turn of the millennium, software problems became headline news with the so-called Year 2000 Problem, or the Y2K bug. The fear was that on January 1, 2000, all hell would break loose, because computers that used two digits for storing years would erroneously assume that a year given as 00 was 1900, when in fact it was 2000. An extremely expensive (and, in retrospect, quite successful) effort to correct these programs had to be taken by software companies worldwide.





## Some examples III

- The software error of a MIM-104 Patriot caused its system clock to drift by one third of a second over a period of one hundred hours – resulting in failure to locate and intercept an incoming Iraqi Al Hussein missile, which then struck Dhahran barracks, Saudi Arabia (February 25, 1991), killing 28 Americans.



## Some examples IV

- ▶ While attempting its first overseas deployment to the Kadena Air Base in Okinawa, Japan, on 11 February 2007, a group of six F-22 Raptors flying from Hickam AFB, Hawaii, experienced multiple computer crashes coincident with their crossing of the 180th meridian of longitude (the International Date Line). The computer failures included at least navigation (completely lost) and communication. The fighters were able to return to Hawaii by following their tankers, something that might have been problematic had the weather not been good. The error was fixed within 48 hours, allowing a delayed deployment.