CSE 20
DISCRETE MATH

Fall 2020

http://cseweb.ucsd.edu/classes/fa20/cse20-a/

Today's learning goals

- Define multiple ways for representing numbers
- Compute the ranges of numbers that can be represented using a given definition

Base b expansion of n

Rosen p. 246

Definition (Rosen p. 246) For b an integer greater than 1 and n a positive integer, the **base b expansion of n** is

$$(a_{k-1} \cdots a_1 a_0)_b$$

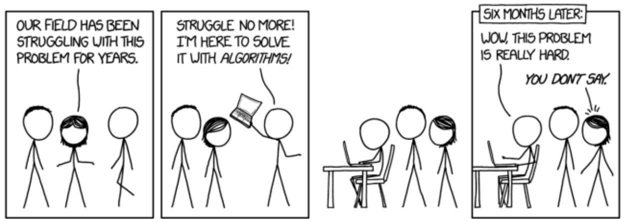
where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

Which of the following is **NOT** correct?

A. $(1)_2 = (1)_8$
B. $(20)_{10} = (10100)_2$
C. $(142)_{10} = (142)_{16}$
D. $(35)_8 = (1D)_{16}$
E. More than one of the above

Converting between bases


OUR FIELD HAS BEEN STRUGGLING WITH THIS PROBLEM FOR YEARS.
STRUGGLE NO MORE! I'M HERE TO SOLVE IT WITH *ALGORITHMS!*
SIX MONTHS LATER:
WOW, THIS PROBLEM IS REALLY HARD.
YOU DON'T SAY.

xkcd

Converting from base b_1 expansion to base b_2 expansion

Algorithm

Representing more

- Base b expansions can express any **positive integers**

- What about
  - Zero?
  - negative integers?
  - rational numbers?
  - other real numbers?

Computer memory

Memory addresses organize storage into fixed-width blocks.

To represent numbers, modify binary to allow for *leading zeros.*

The number 20 can be written using

A. Binary
B. Binary fixed-width 10
C. Binary fixed-width 7
D. Binary fixed-width 4
E. All but one of the above

Computer memory

Memory addresses organize storage into fixed-width blocks.

To represent numbers, modify binary to allow for *leading zeros.*

Definition For b an integer greater than 1, w a positive integer, and n a nonnegative integer ________________,
the **base** b **fixed-width** w **expansion of** n is

$$(a_{w-1} \cdots a_1 a_0)_{b,w}$$

where $a_0, a_1, \ldots, a_{w-1}$ are nonnegative integers less than b and

$$n = a_{w-1} b^{w-1} + \cdots + a_1 b + a_0$$

Binary vs. fixed-width binary

Are there any numbers that can be represented in fixed-width binary that can’t be represented in binary?

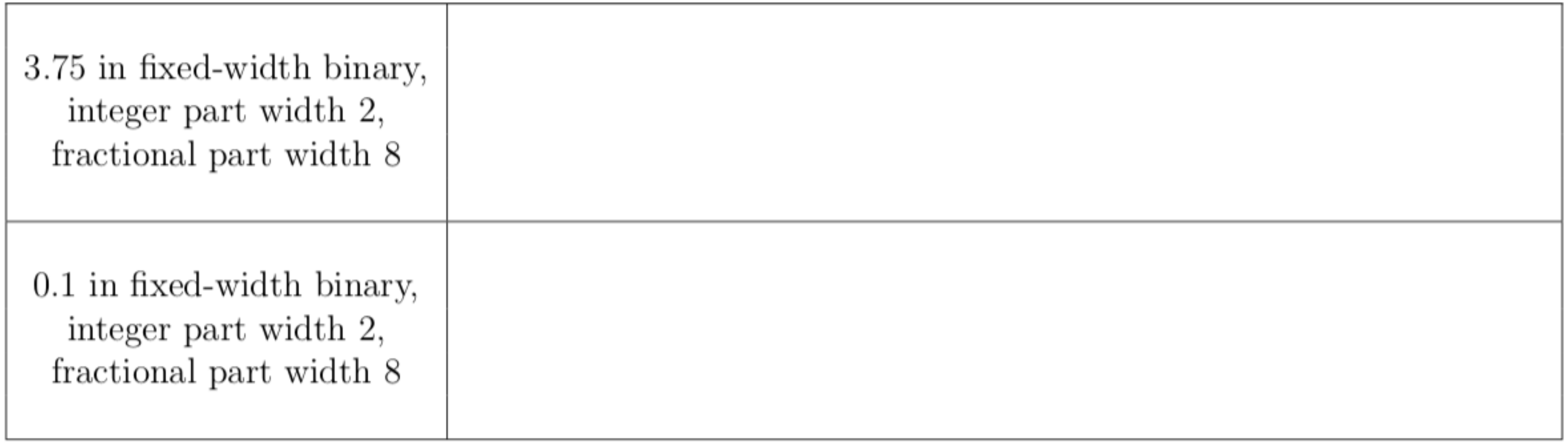
A. Yes, there is one such number.
B. Yes, there are lots of examples.
C. No, because binary representations can get arbitrarily long.
D. No, because they’re both representing whole numbers.
E. I don’t know.

Non-whole numbers

Definition For b an integer greater than 1, w a positive integer, w' a positive integer, and x a real number the **base b fixed-width expansion of x with integer part width w and fractional part width w'** is $(a_{w-1} \cdots a_1 a_0 . c_1 \cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and

and

Non-whole numbers



3.75 in fixed-width binary, integer part width 2, fractional part width 8	
0.1 in fixed-width binary, integer part width 2, fractional part width 8	

Non-whole numbers

For b an integer greater than 1, w a positive integer, w' a positive integer, and x a real number the **base b fixed-width expansion of x with integer part width w and fractional part width w'** is $(a_{w-1}\cdots a_1a_0.c_1\cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and

$$x \geq a_{w-1}b^{w-1} + \cdots + a_1b + a_0 + c_1b^{-1} + \cdots + c_{w'}b^{-w'}$$

and

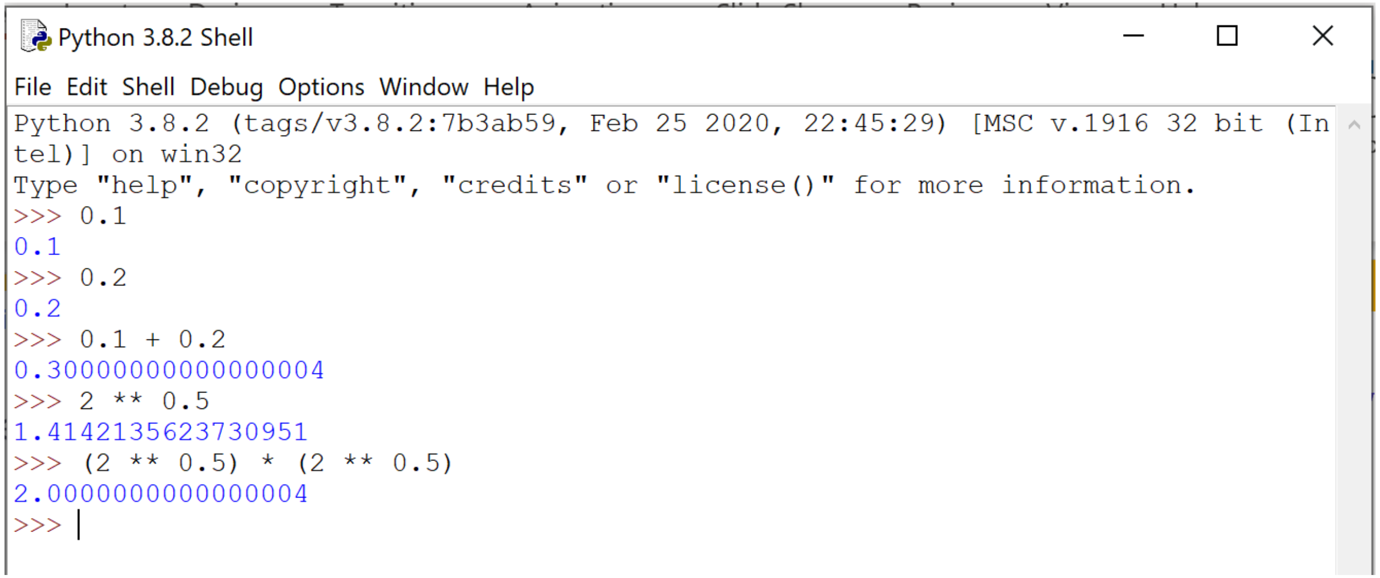
$$x < a_{w-1}b^{w-1} + \cdots + a_1b + a_0 + c_1b^{-1} + \cdots + (c_{w'}+1)b^{-w'}$$

Demo: arithmetic with non-whole numbers

Lets use python!

Note: Python uses floating point, not fixed width representation, but similar rounding errors appear in both.

Demo: arithmetic with non-whole numbers

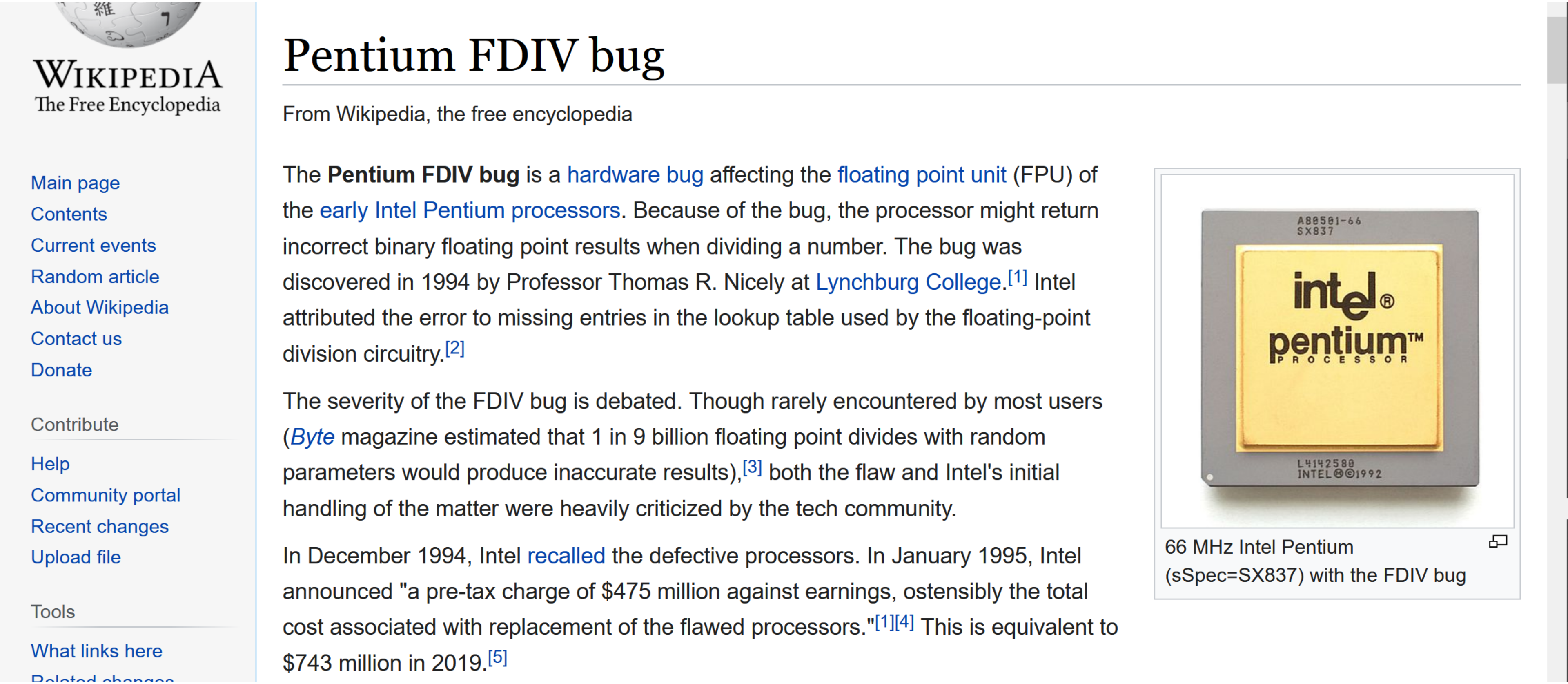


```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 0.1
0.1
>>> 0.2
0.2
>>> 0.1 + 0.2
0.30000000000000004
>>> 2 ** 0.5
1.4142135623730951
>>> (2 ** 0.5) * (2 ** 0.5)
2.0000000000000004
>>> 
```

Note: Python uses floating point, not fixed width representation, but similar rounding errors appear in both.

Professor Peter Coveney, Director of the UCL Centre for Computational Science and study co-author, said: “Our work shows that the behaviour of the chaotic dynamical systems is richer than any digital computer can capture. Chaos is more commonplace than many people may realise and even for very simple chaotic systems, numbers used by digital computers can lead to errors that are not obvious but can have a big impact. Ultimately, computers can’t simulate everything.”

Article: https://www.ucl.ac.uk/news/2019/sep/numbers-limit-how-accurately-digital-computers-model-chaos



Wikipedia
The Free Encyclopedia

- Main page
- Contents
- Current events
- Random article
- About Wikipedia
- Contact us
- Donate

Contribute

- Help
- Community portal
- Recent changes
- Upload file

Tools

- What links here

Pentium FDIV bug

From Wikipedia, the free encyclopedia

The **Pentium FDIV bug** is a hardware bug affecting the floating point unit (FPU) of the early Intel Pentium processors. Because of the bug, the processor might return incorrect binary floating point results when dividing a number. The bug was discovered in 1994 by Professor Thomas R. Nicely at Lynchburg College.[1] Intel attributed the error to missing entries in the lookup table used by the floating-point division circuitry.[2]

66 MHz Intel Pentium (sSpec=SX837) with the FDIV bug

The severity of the FDIV bug is debated. Though rarely encountered by most users (*Byte* magazine estimated that 1 in 9 billion floating point divides with random parameters would produce inaccurate results),[3] both the flaw and Intel's initial handling of the matter were heavily criticized by the tech community.

In December 1994, Intel recalled the defective processors. In January 1995, Intel announced "a pre-tax charge of $475 million against earnings, ostensibly the total cost associated with replacement of the flawed processors."[1][4] This is equivalent to $743 million in 2019.[5]

For next time

- Read website carefully

http://cseweb.ucsd.edu/classes/fa20/cse20-a/

- Homework 1 due on Monday

- Next pre-class reading:
  - Section 4.2 definition of One's, Two's complement (p. 256).
  - Section 1.2 definition of logic gates and circuits (pp. 20-21).