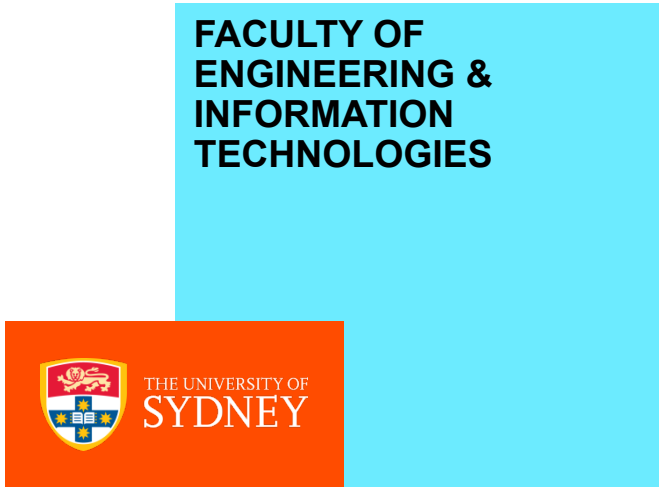
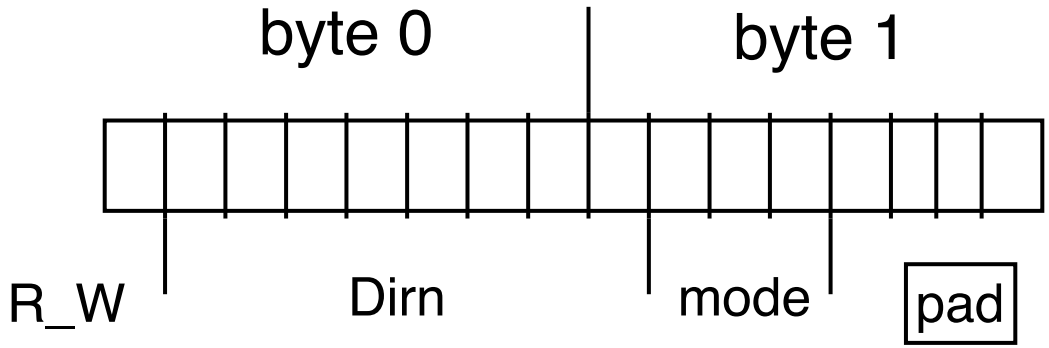
# Bitfields


FACULTY OF
ENGINEERING &
INFORMATION
TECHNOLOGIES
THE UNIVERSITY OF
SYDNEY

# Bit fields

- for some specialised applications you need data fields that are smaller than a byte or are packed into several bytes



# Bit Fields

- can specify a size, in bits, for elements of a structure
- the size is placed after the field name, with a colon between:

```
struct IOdev
{
        unsigned R_W: 1;
        unsigned Dirn: 8;
        unsigned mode: 3;
};
```

**this variable occupies only 3 bits**

```
struct IOdev
{
        unsigned R_W: 1;
        unsigned Dirn: 8;
        unsigned mode: 3;
        unsigned pad: 4;
};

struct IOdev    dev = {1, 0, 7};

main()
{
        printf("mode = %d\n", dev.mode);
}
```

THE UNIVERSITY OF SYDNEY

# Bit Fields

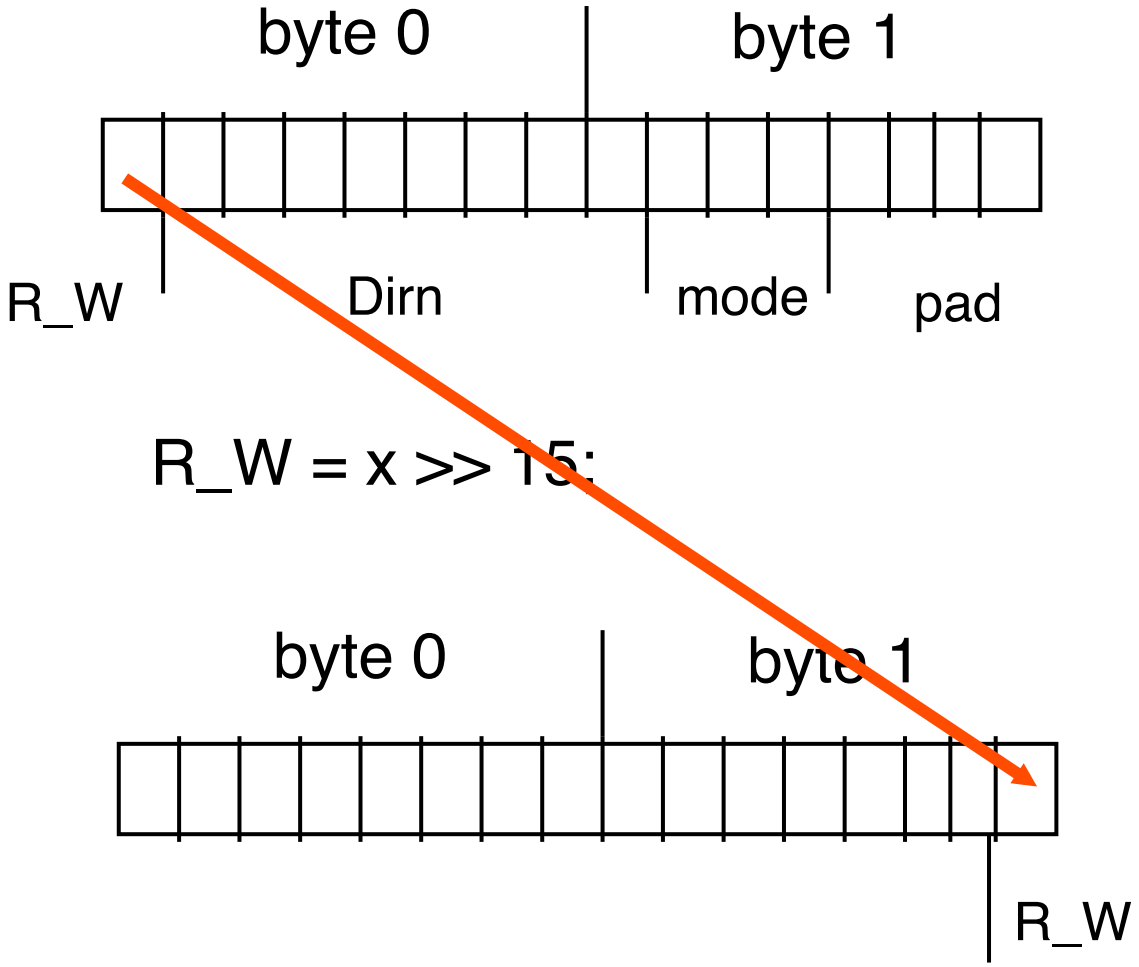
- bitfields are good for low level programming of device registers (drivers, embedded systems etc)
- bitfields are good for “unpacking” data structures
- however: bitfields may not be portable
  - padding
  - left-right vs right-left
- only for experts!

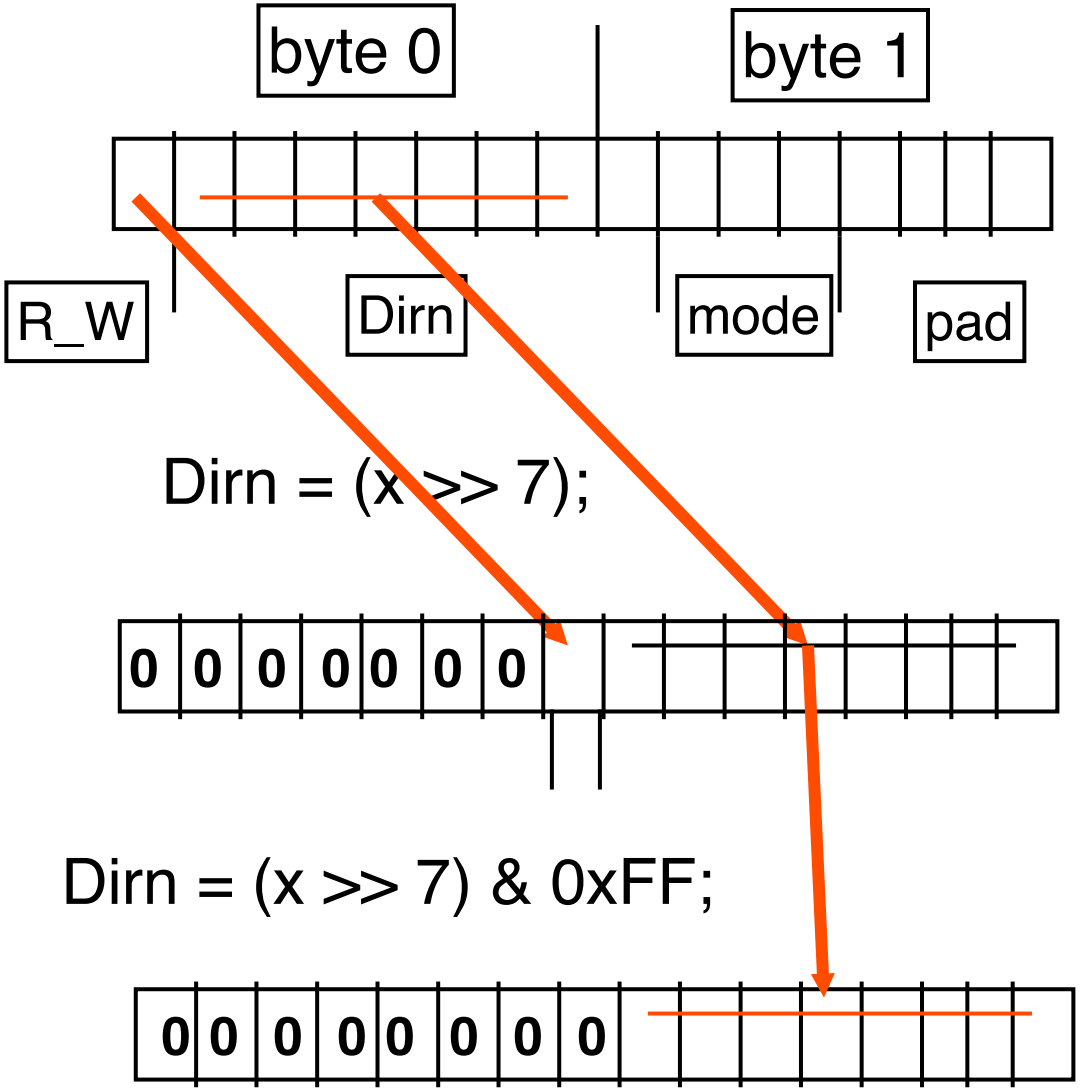
# Bit Fields

- without using the C bitfield syntax you can still unpack bit fields from data
- use shift and logical operations
- eg assuming previous packing of R_W etc:

```
unsigned short x; /* R_W:1, Dirn:8, mode:3, pad:4 */

R_W = x >> 15;
Dirn = (x >>7) & 0xFF;
mode = (x >> 4) & 0x7;
```


byte 0
byte 1
R_W
Dirn
mode
pad
R_W = x >> 15;
byte 0
byte 1
R_W


byte 0
byte 1
R_W
Dirn
mode
pad
Dirn = (x >> 7);
0 0 0 0 0 0 0
Dirn = (x >> 7) & 0xFF;
0 0 0 0 0 0 0 0

# Bit Operations

- shift right: >>
- shift left: <<
- bitwise AND:  &
- bitwise OR: |
- bitwise XOR: ^
- bitwise NOT: ~

    Not to be confused with logical NOT !

Beware signed

THE UNIVERSITY OF SYDNEY

# Summary

- bitfields: easy packing/unpacking of short bit fields
- bit operations: shifting and logical

# Fearful Floats

## Problems for programmers

## IEEE-754 Floating Point representation

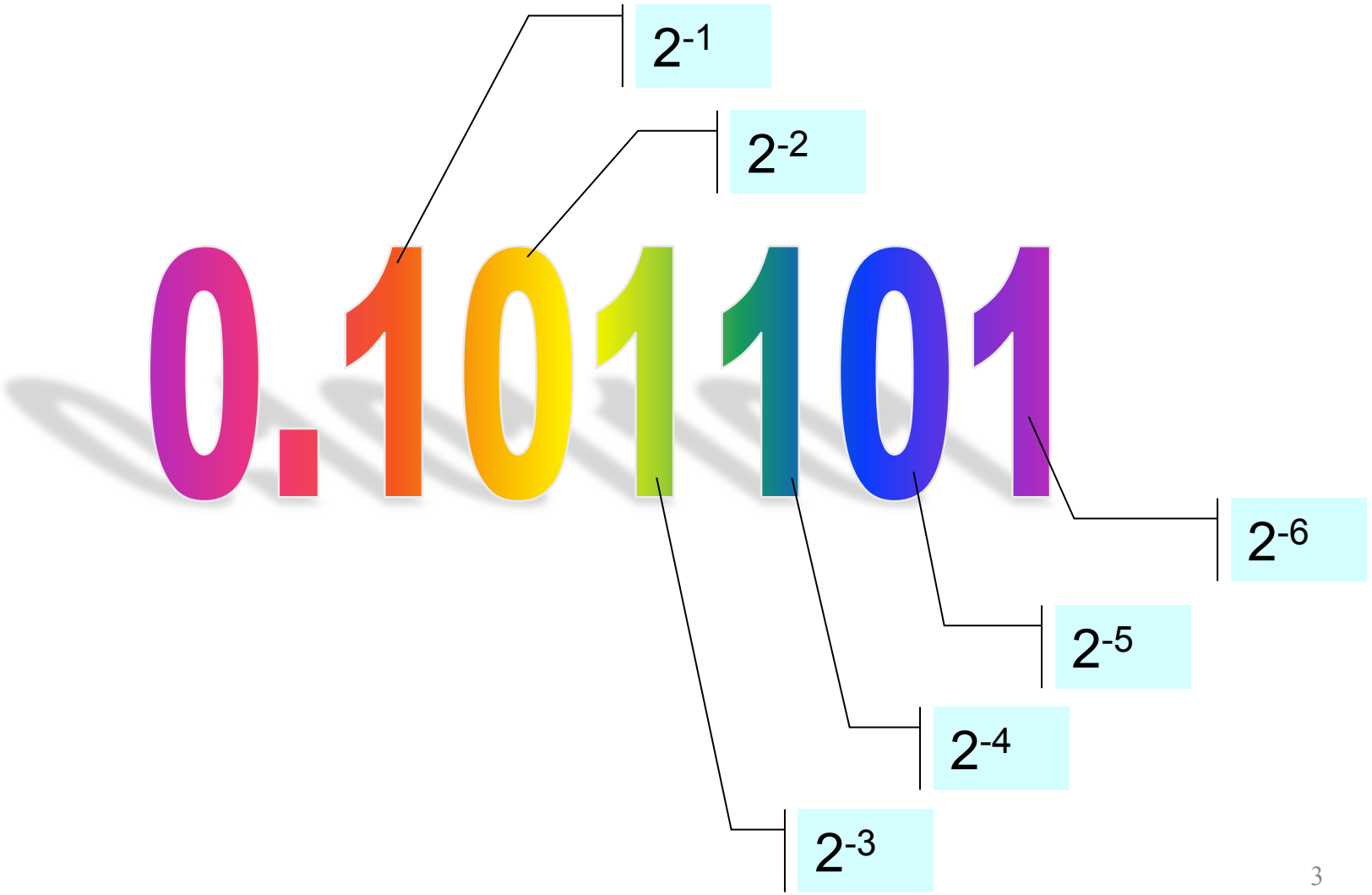
Lecture prepared by Tony Greening and Judy Kay,
and delivered by Bob Kummerfeld

# Fractional representation

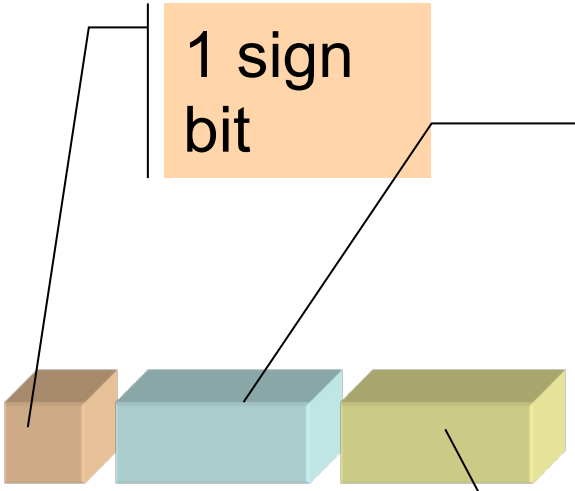
The task:

Use **6** bits to represent fractional numbers using binary representation.

# Binary to decimal



# The “floating point” approach




This is the approach used in “scientific notation”. Thus, in decimal, 315.82 could be expressed as $3.1582 \times 10^2$.

mantissa = 3.1582

exponent = 2

# IEEE-754 Floating Point standard

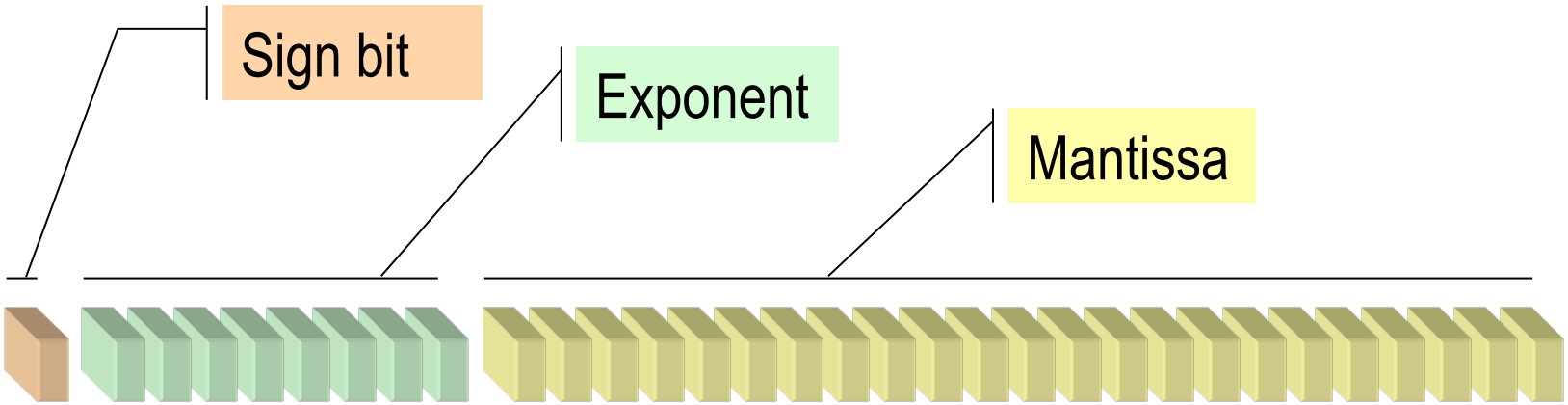
We now look at the details of floating point representation

– Using the IEEE-754 standard

# General structure

Floating point widths are commonly referred to as:

- single-precision 32b
- double-precision 64b
- extended-precision 80b



# Normalisation

Normalisation is a convention that leaves the mantissa in the form:

**1.xxxxx**

where 'x' represents a binary digit.

Thus,

**1101.101**

normalises to

**$1.101101000 \times 2^3$**

# Representation of the sign bit

- A single, leading sign-bit is used:
  - 0 represents positive
  - 1 represents negative

# Representation of the exponent
## Excess notation

- The exponent is represented using *excess* (or *bias*) notation.
  - Specifically, for 32-bit IEEE-754, *excess-127 notation*

- Excess-127 simply means you add 127 to the actual (signed) integer before conversion to binary
  - Then write the number as an (unsigned) integer

# Representation of the exponent (Activity)

Represent the following numbers in excess-127 notation, using an 8-bit data width

**-127**

**0**

**128**

# De-normalised numbers

An exponent field that is filled with zeros denotes a denormalised number.

This means that the assumption of a leading ‘1’ in the mantissa does not apply

Apart from other things, this enables the value ‘0.0’ to be represented...

# Infinity

An exponent field that is filled with ones, together with a mantissa field that is filled with zeros denoted *infinity*.

Note the use of the sign bit allows both *positive* and *negative* infinity to be defined.

# Design considerations

## Range

- It is desirable for the representation method to include as wide a range (maximum - minimum) as possible

# Design considerations

## Density

- It is not possible to represent all numbers. It is desirable to have as high a density of computer-representable numbers as possible.
- To make the point, consider the numbers able to be represented by the following *fixed-point* system:


The same situation applies in floating-point, with a clustering effect instead of periodic.

Representable numbers:

7.75, 7.5, 7.25, 7.0,
6.75, 6.5, 6.25, 6.0,
etc., etc.,
0.75, 0.5, 0.25, 0.0,
-0.25, -0.5, -0.75,
etc., etc.,
-7.0, -7.25, -7.5, -7.75

# Calculations in the “comfort zone”

Sqrt(3) squared = **?**

**3**

0.1 * 10 = **?**

**1.0**

20! = **?**

**“a really big number”**

*(something around 2.432902008176640e+018)*

**BUT in the “real world” of computing, these pleasantries may not hold...**

What if we started with zero (in floating point representation) and then added 0.1 to it 1000 times?

What would the result be?

# What does this print?

```
float   x = 0.1;

        printf("%10.1f\n", x);
        printf("%10.5f\n", x);
        printf("%20.15f\n", x);
        printf("%50.45f\n", x);
```

# Catastrophic cancellation

- Two operands are close to each other
- They cancel out because of order of the operation, eg
  - bignum - bignum + smallnum
  - Quadratic equation discriminant

# Catastrophic cancellation

b = 3.34
a = 1.22
c = 2.28

exact (b * b) - 4 * a * c
.0292

(b * b) rounds to 11.2
4ac to 11,1
answer 0.1

# Benign cancellation

- Avoid the problems by reorganising the expression
- Example: catastrophic cancellation
  - (x * x) - (y * y) has catastrophic
  - Becomes benign (more accurate) as

    (x - y)(x + y)

# About errors

- Sources
  - The raw data eg 7.3
  - Propagation error eg 7.3 + 8.15
  - Representational errors (finiteness)
- Awareness
  - Significant figures
  - Printing
  - calculations
- Measures
  - Absolute error
  - Relative error

# Software errors are everywhere…

Let’s look at some which occur due to the effects of floating point data representation…

# Software errors are everywhere…

- In 1979, NORAD defense radar misinterpreted the moon as an incoming missile…
- In 1983, a software “bug” resulted in an F-14 flying off the edge of an aircraft carrier and into the North Sea.
- In 1984, a 180-degree error caused a Soviet test missile to head towards Hamburg instead of the Arctic.
- The splashdown point of Gemini V was off by over 100 miles because the guidance system neglected movement of the Earth around the Sun.

# June 4, 1996
# Ariane 5 is launched


ARIANESPACE

# Oops! An $(US)8-billion fireworks display!

Ariane 5 exploded in its 37th second of flight. Its payload of 4 satellites was uninsured (as you do!).

It was later found to be due to a small data-representation problem...

- Computed horizontal velocity as floating point number
  - Converted to 16-bit integer
    - Worked OK for Ariane 4
    - Overflowed for Ariane 5
      - Used same software

# February 25, 1991
# The (first) Gulf War

“On February 25th, 1991, an Iraqi Scud hit the barracks in Dahran, Saudi Arabia, killing 28 soldiers from the US Army's 14th Quartermaster detatchment.

**28 soldiers died and over 100 were injured as a result.**

**It was later found to be due to a small data-representation problem...**

A government investigation revealed that the failed intercept at Dhahran had been caused by a software error in the system's clock. The Patriot missile battery at Dhahran had been in operation for 100 hours, by which time the system's internal clock had drifted by one third of a second. Due to the closure speed of the interceptor and the target, this resulted in a miss distance of 600 meters.

# Summary

- Appreciation of floating point problems:
    - big disasters may result from small errors
    - small errors are hard to find
- Common pitfalls to be aware of
    - Expect that floats may be inaccurate: the 0.1 problem due to 0.1 being between two representable floats
    - the out of range problem - number too big or too small
- Never, ever test floats for equality

end of segment

The radar system had successfully detected the Scud and predicted where to look for it next, but because of the time error, looked in the wrong part of the sky and found no missile. With no missile, the initial detection was assumed to be a spurious track and the missile was removed from the system. No interception was attempted, and the missile impacted on a barracks killing 28 soldiers.

At the time, the Israelis had already identified the problem and informed the US Army and the PATRIOT Project Office (the software manufacturer) on February 11th, 1991, but no upgrade was present at the time. As a stopgap measure, the Israelis recommended rebooting the system's computers regularly, however, Army officials did not understand how often they needed to do so. The manufacturer supplied updated software to the Army on February 26th, the day after the Scud struck the Army barracks.

# 1982+
# The Vancouver Stock Exchange

# The Vancouver STX goes psycho (for 2 years!)

**Question:**

How did a Vancouver Stock Exchange index gain around 574 points while the stock prices were unchanged?

**Partial Answer:**

In 1982 the Vancouver STX introduced an index with an initial value of 1000.000

After 22 months the index was listed as 1524.881

*But this was incorrect!* Its *true* value was 1098.811

**It was later found to be due to a small data-representation problem...**

# Sources

- David Goldberg, What Every Computer Scientist Should Know about Floating-Point Arithmetic, covers important aspects of floating-point arithmetic from the perspective of IEEE 754 from the IEEE web site.
- Kernighan and Pike, Practice of Programming.
- Ariane 5 video
  - http://www.estec.esa.nl/spdwww/cluster/restored/pics/ar5lnch.mpg
- Ariane 5 explosion video (converted from qt format to MPEG by ye trusty ‘ol SGI O2):
  - http://graffiti.cribx1.u-bordeaux.fr/roussel/anim-e_07f.shtml
- ariane 5 explosion photo
  - http://www.mpia-hd.mpg.de/SUW/SuW/1996/10-96/S713Abb1.html
- STX photo:
  - http://www.abc.net.au/news/imageLibrary/00/04/18/g00041820000418stocks.jpg
- Gulf war photo:
  - http://www.washingtonpost.com/wp-srv/inatl/images/fogofwarpics/analysis/bombing1.jpg
- Patriot missile photo:
  - http://www.ima.umn.edu/~arnold/disasters/patriot.html

end of segment

# Representation of the exponent (Activity)

Represent the following numbers in excess-127 notation, using an 8-bit data width

| | | |
|---|---|---|
| **-127** | 0 | `00000000` |
| **0** | 127 | `01111111` |
| **128** | 255 | `11111111` |