Representations 2

Prof. David August

COS 217

Today

- Unsigned Multiplication
- Fixed Point
- Floating Point

Multiplication

Computing Exact Product of w-bit numbers x, y

- Need 2w bits

Unsigned: $0 \le x * y \le (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$

Two's Complement:

min: $x * y \ge (-2^{w-1})(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$

max: $x * y \le (-2^{w-1})^2 = 2^{2w-2}$

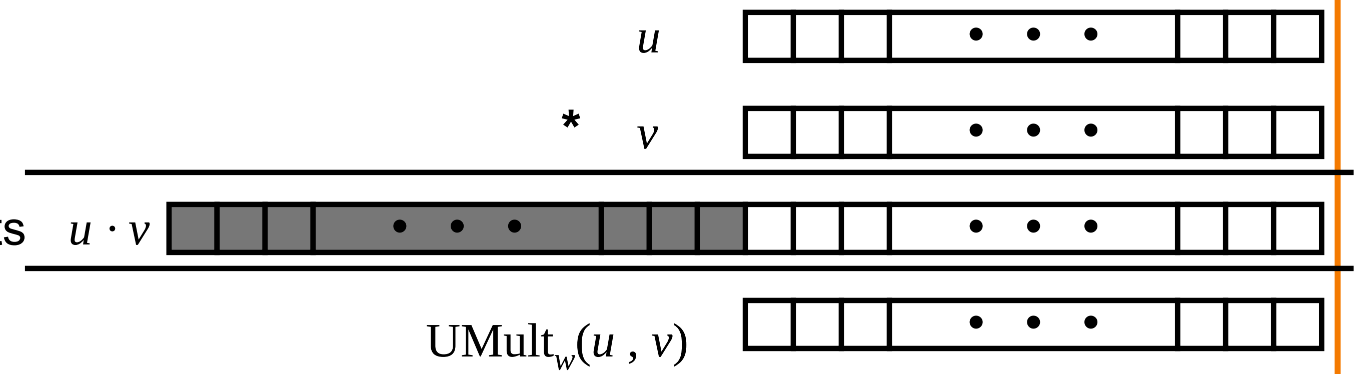
- Maintaining Exact Results
  - Need unbounded representation size
  - Done in software by *arbitrary precision* arithmetic packages
  - Also implemented in Lisp, ML, and other languages

Unsigned Multiplication in C

Operands: *w* bits

True Product: 2**w* bits

Discard *w* bits: *w* bits



- Standard Multiplication Function
  - Ignores high order w bits
- Implements Modular Arithmetic
  - $\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$
- What about unsigned integer division?

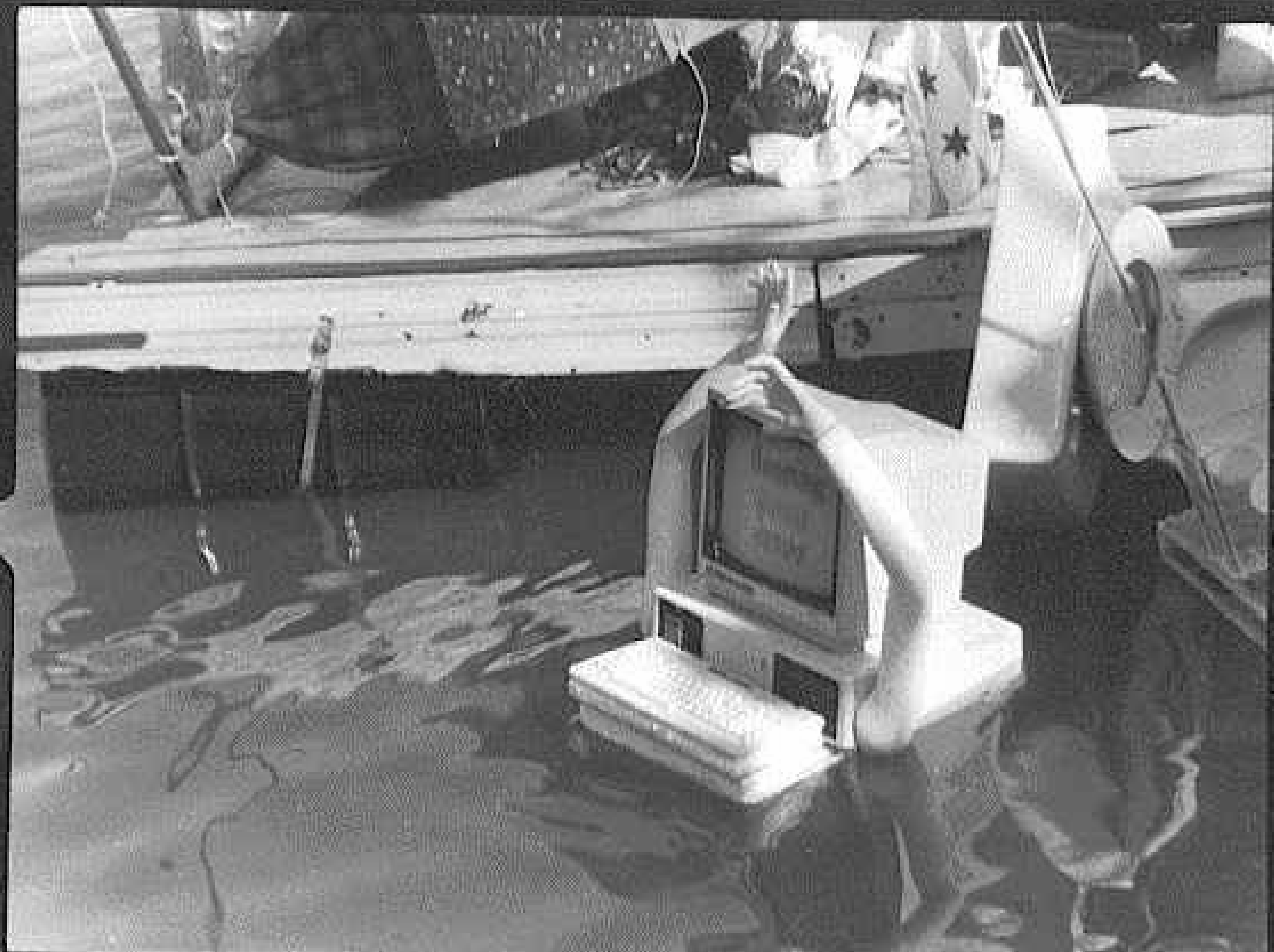
Unsigned Multiplication

Binary makes it easy:

- 0 => place 0 (0 x multiplicand)
- 1 => place a copy (1 x multiplicand)

Key sub-parts:

- Place a copy or not
- Shift copies appropriately
- Final addition



Representations

What can be represented in N bits?

Unsigned: $0 \rightarrow 2^n - 1$

Signed: $-2^{n-1} \rightarrow 2^{n-1} - 1$

What about:

Very large numbers?	9,349,787,762,244,859,087,678
Very small numbers?	0.0000000000000000000004691
Rationals?	2/3
Irrationals?	SQRT(2)
Transcendentals?	e, PI

Interpretations

Bit Pattern	Method 1	Method 2	Method 3
000	0	0	0
001	1	1	0.1
010	e	2	0.2
011	pi	4	0.3
100	4	8	0.4
101	-pi	16	0.5
110	-e	32	0.6
111	-1	64	0.7

What should we do? Another method?

The Binary Point

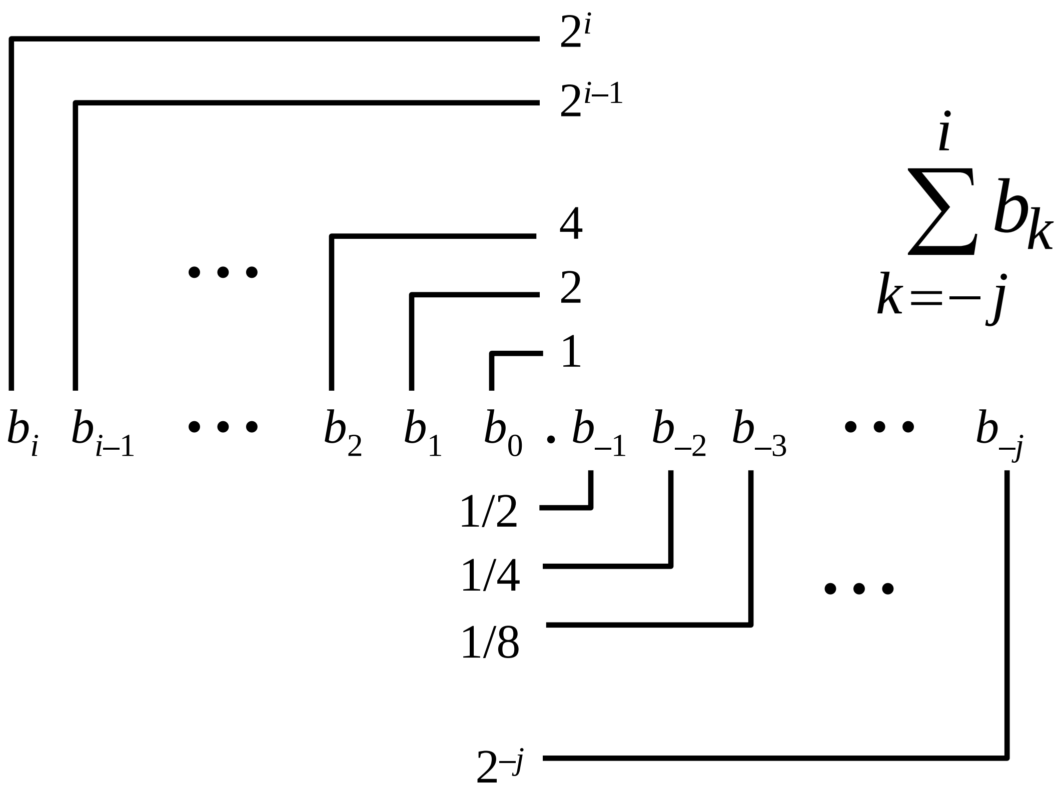
$101.11_2 = 4 + 1 + ½ + ¼ = 5.75$

Observations:

- Divide by 2 by shifting point left
- $0.111111\ldots_2$ is just below 1.0
- Some numbers cannot be exactly represented well

  $1/10 \rightarrow 0.000110011[0011]^*\ldots_2$

Obvious Approach: Fixed Point



$$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Fixed Point

In w-bits (w = i + j):

- use i-bits for left of binary point
- use j-bits for right of binary point

Qualities:

- Easy to understand
- Arithmetic relatively easy to implement…
- Precision and Magnitude:

16-bits, i=j=8: 0 → 255.99609375

Step size: 0.00390625

Another Approach: Scientific Notation

decimal point

Exponent
Sign, magnitude

$$6.02 \times 10^{23}$$

Mantissa
Sign, magnitude

radix (base)

- In Binary:

$$\text{radix} = 2$$

$$\text{value} = (-1)^s \times M \times 2^E$$

s	E	M

- How is this better than fixed point?

IEEE 754 Floating Point

- Established in 1980 as uniform standard for floating point arithmetic
- Supported by all major CPUs
- In 99.999% of all machines used today

Driven by Numerical Concerns

- Standards for rounding, overflow, underflow
- Primarily numerical analysts rather than hardware types defined standard

This is where it gets a little involved…

IEEE 754 Floating Point Standard

- Single precision: 8 bit exponent, 23 bit significand
- Double precision: 11 bit exponent, 52 bit significand

- Significand *M* normally in range [1.0,2.0) → Imply 1
- Exponent *E* biased exponent → B is bias ($B = 2^{|E|-1} - 1$)

$$N = (-1)^s \times 1.M \times 2^{E - B}$$

s	E	M

- Bias allows integer comparison (almost)!

  0000…0000 is most negative exponent

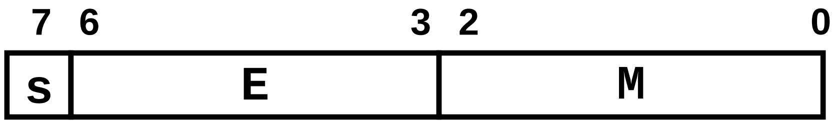
  1111…1111 is most positive exponent

IEEE 754 Floating Point Example

Define Wimpy Precision as:

1 sign bit, 4 bit exponent, 3 bit significand, B = 7

Represent: -0.75



IEEE 754 Floating Point
There's more!

Normalized: E ≠ 000…0 and E ≠ 111…1

- Recall the implied 1.*xxxxx*

Special Values: E = 111…1

- M = 000…0:
  - Represents +/- ∞ (infinity)
  - Used in overflow
  - Examples: 1.0/0.0 = +∞, 1.0/–0.0 = –∞
  - Further computations with infinity possible
  - Example: X/0 > Y may be a valid comparison

IEEE 754 Special Exponents

Normalized: E ≠ 000…0 and E ≠ 111…1

Special Values: E = 111…1

- M ≠ 000…0:
  - Not-a-Number (NaN)
  - Represents invalid numeric value or operation
  - Not a number, but not infinity (e.q. sqrt(-4))
  - Examples: sqrt(–1), ∞ – ∞
  - NaNs propagate: f(NaN) = NaN

IEEE 754 Special Exponents

Normalized: E ≠ 000…0 and E ≠ 111…1

- Recall the implied 1.*xxxxx*

Denormalized: E = 000…0

- M = 000…0
  - Represents value 0
  - Note the distinct values +0 and –0

IEEE 754 Special Exponents

Normalized: E ≠ 000…0 and E ≠ 111…1

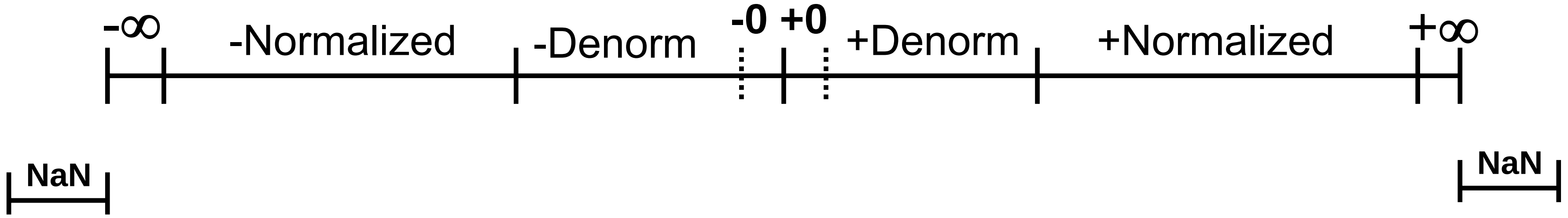
- Recall the implied 1.*xxxxx*

Denormalized: E = 000…0

- M ≠ 000…0
  - Numbers very close to 0.0
  - Lose precision as magnitude gets smaller
  - “Gradual underflow”

Exponent	–*Bias* + 1
Significand	$\texttt{0.xxx...x}_2$

Encoding Map



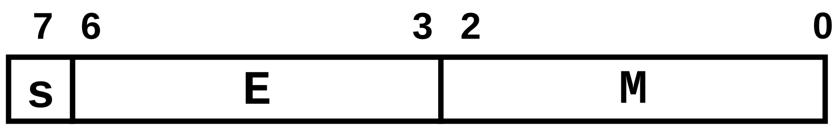
Dynamic Range

	S E M	exp	value	
Denormalized numbers	0 0000 000	n/a	0	
	0 0000 001	-6	1/512	← closest to zero
	0 0000 010	-6	2/512	
	…			
	0 0000 110	-6	6/512	
	0 0000 111	-6	7/512	← largest denorm
Normalized numbers	0 0001 000	-6	8/512	← smallest norm
	0 0001 001	-6	9/512	
	…			
	0 0110 110	-1	28/32	
	0 0110 111	-1	30/32	← closest to 1 below
	0 0111 000	0	1	
	0 0111 001	0	36/32	← closest to 1 above
	0 0111 010	0	40/32	
	…			
	0 1110 110	7	224	
	0 1110 111	7	240	← largest norm
	0 1111 000	n/a	inf	

Wimpy Precision

Define Wimpy Precision as:

1 sign bit, 4 bit exponent, 3 bit significand, B = 7



E = 1-14: Normalized

E = 0: Denormalized

E = 15: Infinity/ NaN