

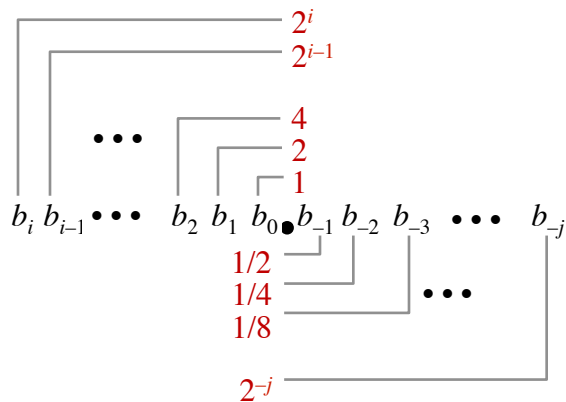
CSci 370: Computer Organization

Floating Point Representation

Instructor:

Professor Stephen P. Carl

Fractional Binary Numbers



Representation

Bits to right of “binary point” represent fractional powers of 2

This represents the rational number given by:

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

Fractional binary numbers

What is 1011.101?

Fractional Binary Numbers: Examples

Value	Representation
$5 \frac{3}{4}$	101.11_2
$2 \frac{7}{8}$	10.111_2
$\frac{63}{64}$	0.111111_2

Observations

To divide by 2, shift bits right

To multiply by 2, shift bits left

Numbers of the form $0.111111\dots_2$ are just below 1.0

- $1/2 + 1/4 + 1/8 + \dots + 1/2^i + \dots \rightarrow 1.0$
- Use notation $1.0 - \epsilon$

Representable Numbers

Floating point representation

- Can only *exactly* represent numbers of the form $x/2^k$
- Other rational numbers have repeating bit representations:

Value	Representation
1/3	0.0101010101 [01] ₂ ...
1/5	0.001100110011 [0011] ₂ ...
1/10	0.0001100110011 [0011] ₂ ...

IEEE Floating Point Standards

- **The IEEE Standard 754**

Was established in 1985 as uniform standard for floating point arithmetic
 (Before that, many idiosyncratic formats)
 Now supported by all major CPUs

- **Driven by numerical concerns**

Nice standards for rounding, overflow, and underflow
 But: hard to make fast in hardware - numerical analysts
 predominated over hardware designers in defining standard

Floating Point Representation

Numerical Form:

$$(-1)^s M 2^E$$

Sign bit s determines whether number is negative or positive

Significand M normally a fractional value in range [1.0,2.0).

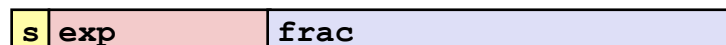
Exponent E weights value by power of two

Encoding:

MSB s is sign bit s

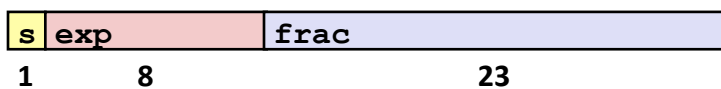
exp field encodes E (but is not equal to E)

frac field encodes M (but is not equal to M)

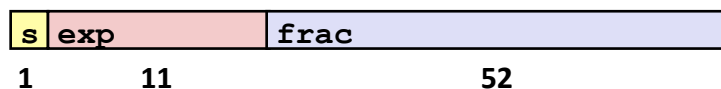


Precisions

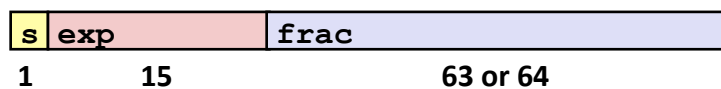
Single precision: 32 bits



Double precision: 64 bits



Extended precision: 80 bits (Intel only)



Normalized Values

- **Condition:** $\text{exp} \neq 000\dots 0$ and $\text{exp} \neq 111\dots 1$

- **Exponent coded as *biased* value:** $E = \text{Exp} - \text{Bias}$
Exp: unsigned value exp
Bias = $2^{e-1} - 1$, where e is number of exponent bits
 - Single precision: 127 (*Exp*: 1...254, *E*: -126...127)
 - Double precision: 1023 (*Exp*: 1...2046, *E*: -1022...1023)

- **Significand coded with implied leading 1:** $M = 1.\text{xxx}\dots\text{x}_2$
 $\text{xxx}\dots\text{x}$: bits of *frac*
 Minimum when $000\dots 0$ ($M = 1.0$)
 Maximum when $111\dots 1$ ($M = 2.0 - \epsilon$)
 Get extra leading bit for "free"

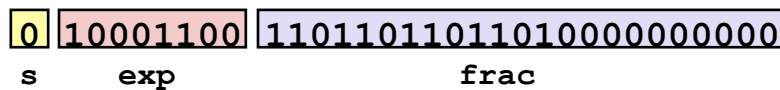
Normalized Encoding Example

- **Value:** float $F = 15213.0$;
 $15213_{10} = 11101101101101_2$
 $= 1.1101101101101_2 \times 2^{13}$

- **Significand**
 $M = 1.\underline{1101101101101}_2$
 $\text{frac} = \underline{1101101101101}0000000000_2$

- **Exponent**
 $E = 13$
 $\text{Bias} = 127$
 $\text{Exp} = 140 = 10001100_2$

- **Result:**



Denormalized Values

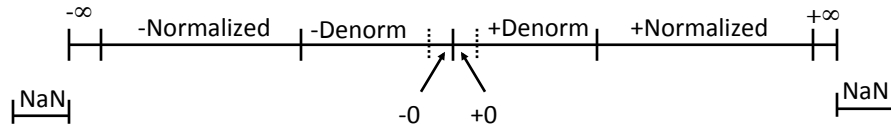
- **Condition:** $\text{exp} = 000\dots0$
- **Exponent value:** $E = -\text{Bias} + 1$ (instead of $E = 0 - \text{Bias}$)
- **Significand coded with implied leading 0:** $M = 0 . \text{xxx}\dots\text{x}_2$
 - $\text{xxx}\dots\text{x}$: bits of frac
- **Cases**
 - $\text{exp} = 000\dots0, \text{frac} = 000\dots0$
 - Represents value 0
 - Note distinct values: +0 and -0
 - $\text{exp} = 000\dots0, \text{frac} \neq 000\dots0$
 - Numbers very close to 0.0
 - You lose precision as numbers get smaller
 - Equispaced on floating point “number line”

Special Values

Condition: $\text{exp} = 111\dots1$

- **Case:** $\text{exp} = 111\dots1, \text{frac} = 000\dots0$
 - Represents value ∞ (infinity)
 - Result of operation that overflows
 - Both positive and negative infinity
 - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
- **Case:** $\text{exp} = 111\dots1, \text{frac} \neq 000\dots0$
 - Not-a-Number (NaN)
 - Represents case when no numeric value can be determined
 - E.g., $\text{sqrt}(-1), \infty - \infty, \infty * 0$

Visualization: Floating Point Encodings



Interesting Numbers

{single precision}

Description	exp	frac	Numeric Value
Zero	00...00	00...00	0.0
Smallest Pos. Denorm. Single $\approx 1.4 \times 10^{-45}$	00...00	00...01	$2^{-23} \times 2^{-126}$
Largest Denormalized Single $\approx 1.18 \times 10^{-38}$	00...00	11...11	$(1.0 - \epsilon) \times 2^{-126}$
Smallest Pos. Normalized Just larger than largest denormalized	00...01	00...00	1.0×2^{-126}
One	01...11	00...00	1.0
Largest Normalized Single $\approx 3.4 \times 10^{38}$	11...10	11...11	$(2.0 - \epsilon) \times 2^{127}$

Interesting Numbers

{double precision}

Description	exp	frac	Numeric Value
Zero	00...00	00...00	0.0
Smallest Pos. Denorm. Double $\approx 4.9 \times 10^{-324}$	00...00	00...01	$2^{-52} \times 2^{-126}$
Largest Denormalized Double $\approx 2.2 \times 10^{-308}$	00...00	11...11	$(1.0 - \epsilon) \times 2^{-1022}$
Smallest Pos. Normalized ▪ Just larger than largest denormalized	00...01	00...00	1.0×2^{-1022}
One	01...11	00...00	1.0
Largest Normalized ▪ Double $\approx 1.8 \times 10^{308}$	11...10	11...11	$(2.0 - \epsilon) \times 2^{1023}$

Special Properties of FP Encoding

- **FP Zero Same as Integer Zero**
 - All bits = 0
- **Can (Almost) Use Unsigned Integer Comparison**
 - Must first compare sign bits
 - Must consider $-0 = 0$
 - NaNs problematic
 - Will be greater than any other values
 - What should comparison yield?
 - Otherwise OK
 - Denorm vs. normalized
 - Normalized vs. infinity

Floating Point Operations: Basic Idea

$$x +_f y = \text{Round}(x + y)$$

$$x \times_f y = \text{Round}(x \times y)$$

Basic idea

First, **compute exact result**

Then, *make it fit* into desired precision

- Might overflow if exponent too large
- Might need to **round to fit into frac**

Rounding

Rounding Modes (illustrate with \$ rounding)

	\$1.40	\$1.60	\$1.50	\$2.50	-\$1.50
Towards zero	\$1	\$1	\$1	\$2	-\$1
Round down ($-\infty$)	\$1	\$1	\$1	\$2	-\$2
Round up ($+\infty$)	\$2	\$2	\$2	\$3	-\$1
Nearest Even (default)	\$1	\$2	\$2	\$2	-\$2

What are the advantages of the modes?

Closer Look at Round-To-Even

■ Default Rounding Mode

- Hard to get any other kind without dropping into assembly
- All others are statistically biased: the sum of the set of positive numbers will consistently be over- or under-estimated

■ Applying to Other Decimal Places / Bit Positions

- When exactly halfway between two possible values, round so that *least significant digit is even*

- E.g., round to nearest hundredth

1.2349999	1.23	(Less than half way)
1.2350001	1.24	(Greater than half way)
1.2350000	1.24	(Half way—round up)
1.2450000	1.24	(Half way—round down)

Rounding Binary Numbers

■ Binary Fractional Numbers

“Even” when least significant bit is 0

“Half way” when bits to right of rounding position = 100...₂

■ Examples

Round to nearest 1/4 - 2 bits to right of binary point

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00 011 ₂	10.00 ₂	(<1/2—down)	2
2 3/16	10.00 110 ₂	10.01 ₂	(>1/2—up)	2 1/4
2 7/8	10.11 100 ₂	11.00 ₂	(1/2—up)	3
2 5/8	10.10 100 ₂	10.10 ₂	(1/2—down)	2 1/2

Floating Point Addition

$$(-1)^{s_1} M_1 2^{E_1} + (-1)^{s_2} M_2 2^{E_2}$$

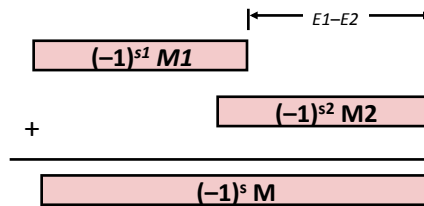
Assume $E_1 > E_2$

■ **Exact Result:** $(-1)^s M 2^E$

Sign s , significand M :

- Result of signed align & add

Exponent E : E_1



■ **Fixing**

If $M \geq 2$, shift M right, increment E

if $M < 1$, shift M left k positions, decrement E by k

Overflow if E out of range

Round M to fit **frac** precision

Mathematical Properties of FP Add

Algebraic properties

- | | |
|--|---------------|
| Closed under addition? | Yes |
| ▪ But may generate infinity or NaN | |
| Commutative? | Yes |
| Associative? | No |
| ▪ Overflow and inexactness of rounding | |
| 0 is additive identity? | Yes |
| Every element has additive inverse | Almost |
| ▪ Except for infinities & NaNs | |

Monotonicity

- | | |
|-------------------------------------|---------------|
| $a \geq b \Rightarrow a+c \geq b+c$ | Almost |
| ▪ Except for infinities & NaNs | |

FP Multiplication

$$(-1)^{s1} M1 2^{E1} \times (-1)^{s2} M2 2^{E2}$$

■ **Exact Result:** $(-1)^s M 2^E$

Sign s : $s1 \wedge s2$
 Significand M : $M1 * M2$
 Exponent E : $E1 + E2$

■ **Fixing**

If $M \geq 2$, shift M right, increment E
 If E out of range, set overflow
 Round M to fit frac precision

■ **Implementation**

Biggest chore is multiplying significands

Mathematical Properties of FP Mult

Algebraic Properties

- Closed under multiplication? **Yes**
 - But may generate infinity or NaN
- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
 - Possibility of overflow, inexactness of rounding
- 1 is multiplicative identity? **Yes**
- Multiplication distributes over addition? **No**
 - Possibility of overflow, inexactness of rounding

Monotonicity

- $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c$ **Almost**
 - Except for infinities & NaNs

Floating Point in C

- **C Guarantees Two Levels**

`float` single precision
`double` double precision

- **Casting between `int`, `float`, and `double` changes bit representation:**

`Double/float` → `int`

- Truncate fractional part, like rounding toward zero
- Not defined when out of range or NaN: Generally sets to TMin

`int` → `double`

- Exact conversion, as long as `int` has ≤ 53 bit word size

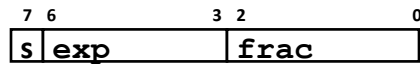
`int` → `float`

- Will round according to rounding mode

Creating Floating Point Numbers

Steps

- 1) Normalize to have leading 1
- 2) Round to fit within fraction
- 3) Postnormalize to deal with effects of rounding

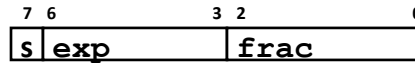


Example: Convert 8-bit unsigned numbers to tiny floating point format

Example Numbers

128	10000000
15	00001101
33	00010001
35	00010011
138	10001010
63	00111111

1) Normalize



To set binary point to get numbers of the form 1.xxxxx,

- adjust all to have leading one
- decrement exponent as shift left

Value	Binary Fraction	Exponent	
128	10000000	1.0000000	7
15	00001101	1.1010000	3
17	00010001	1.0001000	5
19	00010011	1.0011000	5
138	10001010	1.0001010	7
63	00111111	1.1111100	5

2) Rounding



Guard bit: LSB of result

Round bit: 1st bit removed

Sticky bit: OR of remaining bits

Round up conditions

If Round = 1, Sticky = 1 → > 0.5

If Guard = 1, Round = 1, Sticky = 0 → Round to even

Value	Fraction	GRS	Incr?	Rounded
128	1.0000000	000	N	1.000
15	1.1010000	100	N	1.101
17	1.0001000	010	N	1.000
19	1.0011000	110	Y	1.010
138	1.0001010	011	Y	1.001
63	1.1111100	111	Y	10.000

3) Postnormalize

Note:

- Rounding may have caused overflow
- Handle by shifting right once & incrementing exponent

<i>Value</i>	<i>Rounded</i>	<i>Exp</i>	<i>Adjusted</i>	<i>Result</i>
128	1.000	7		128
15	1.101	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000/6	64

Summary

- IEEE Floating Point represents numbers of form $M \times 2^E$
- One can reason about operations independent of implementation
 - As if computed with perfect precision and then rounded
- Not the same as real arithmetic
 - Violates associativity/distributivity
 - Makes life difficult for compilers & programming industrial-strength numerical applications