Floating Point Arithmetic

Computer Systems, Section 2.4

Abstraction

Anything that is not an integer can be thought of as <int>.<decimal>

e.g. 391.1356

Or can be thought of as

<int> + <numerator>/<denominator>

e.g.

391 + 1356/10000

or 201456 13/16

Leak 1

Numbers may not be exactly precise!

$1/3 \; != \; 0.33333333333333333333$

$6.02214129 \times 10^{23}$ is not an exact Avogadro's constant

3.14159265358979323846264338327950288419716939937510

is not exactly π

Leak 2

On computers, fractional numbers must be represented by bits

Implies base 2

Implies “binary point”

	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	…
	4	2	1	.	1/2	1/4	1/8	1/16	…
…	1	0	1	.	1	0	0	1	…

Leak 3

Almost infinite number of ways to represent floating point numbers

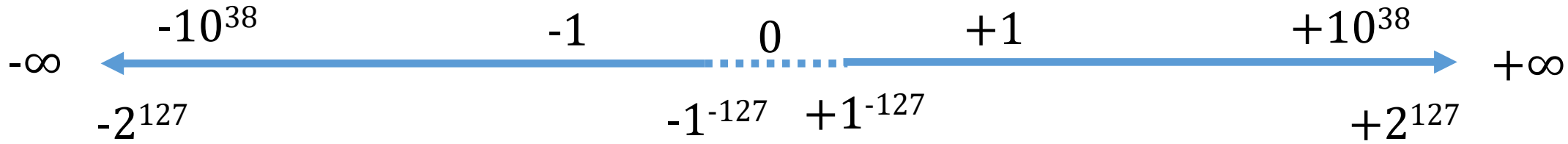
- Implied binary point: 1101 1010 = 1101 10.10 = 54.5
- int numerator, int denominator: 0110 1101 / 0000 00010 = 109/2
- Scientific notation with int integer, int fraction, int exponent
  0000 0101 / 0111 0011 / 0000 0001
  $= (5+1/4+1/8+1/16+1/128+1/256) \times 10^{1} = 54.4921875$
- …

IEEE Standard

- First convert the number to the form:

$$value = -1^{S} \times SIG \times 2^{exp}$$

- S = 0 (positive) or 1 (negative)
- $1 \leq SIG < 2$ (except for "denormal" numbers)
- $-127 \leq \exp \leq 127$



Standard: IEEE 754

- Value Representation:
  - Decimal: [+/-]<digit>.<fraction> x $10^{<exponent>}$ e.g. 6.022 x 10^{23}
  - Binary: [+/-]1.<fraction> x $2^{<exponent>}$ e.g 1.11111110000101… x 2^{78}
  - Special case for 0, +/- ∞ (INFINITY), "Not a Number" (NAN)

- Bit Representation (float)

S	EXP								FRAC																			
b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

IEEE 754 Special Cases

- +/-0

S	EXP								FRAC																			
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- +/-∞ (INFINITY)

S	EXP								FRAC																			
S	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- +/- Not a Number (NAN)

S	EXP								FRAC																			
S	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

IEEE 754 Value vs. Bits

VALUE

- Decimal: -729.6
- Binary: 10 1101 1001.1001 1…
- B/Norm: 1.0110 1100 … x 2^{9}
- Exponent: 9

BITS

- Sign bit 0=+, 1=- : 1
- FRAC: 0110 1100 1100…
- Biased Exponent: 9+127=136 = 1000 1000

IEEE 754 Value vs. Bits

- Sign bit is 0 for positive, 1 for negative… mathematically $(-1)^S$
- Except in special cases, $fraction = 1.FRAC$
- Value must be **normalized** before it can be converted to bits
  - Normalization: moving binary point right of first 1 and adjusting exponent
  - E.g. $0b100110.1010 \times 2^5 = 1.001101010 \times 2^{10}$
  - E.g. $0b0.0001010110 [\times 2^0] = 1.010110 \times 2^{-4}$
- In bit form, exponent is >= 0
  - Abstract exponent value is **biased** – add a constant
  - $EXP = exponent + 127 = exponent + 2^7 - 1 = 0x80 + exponent - 1$
  - $exponent = EXP - 127$ where EXP is 8 bit unsigned binary

value
bits
implicit
bits
value
value
bits

Example: Value to Bits

- Value: 3.1416
  - Convert to binary: 11.001001000011111….
  - Normalize: $1.100100100001111… \times 2^1$
- Bits:
  - $S = 0$ (positive)
  - $EXP = 1 + 127 = 128 = 0b1000\ 0000$
  - $FRAC = 100100100001111….$

See Also: xmp_float

S	EXP								FRAC																			
0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	1
4				0					4			9			0			F			F			9				

- Bits=0x40490FF9

Example: Bits to Value

- Bits: 0x6703ece6

S	EXP								FRAC																			
0	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0
6				7					0				3				E			C				E				6			

  - $S = 0, +$
  - $EXP = 0xCE = 206$, $exponent = 206 - 127 = 79$
  - $fraction = 1.00000111111011\ldots$
- Value: $+\ 0b1.00000111111011..\ x\ 2^{79} = 6.23\ x\ 10^{23}$

"Denormal" or "Subnormal" Numbers

- If EXP bits are zero,
  - we do NOT assume *fraction* starts with 1.<xxx…>
  - we assume it starts with 0.<xxx…>
- Allows numbers smaller than 2^{-126} to be represented
- Smallest Normal: $1.0 \times 2^{-126} = 1.17549435 \times 10^{-38}$

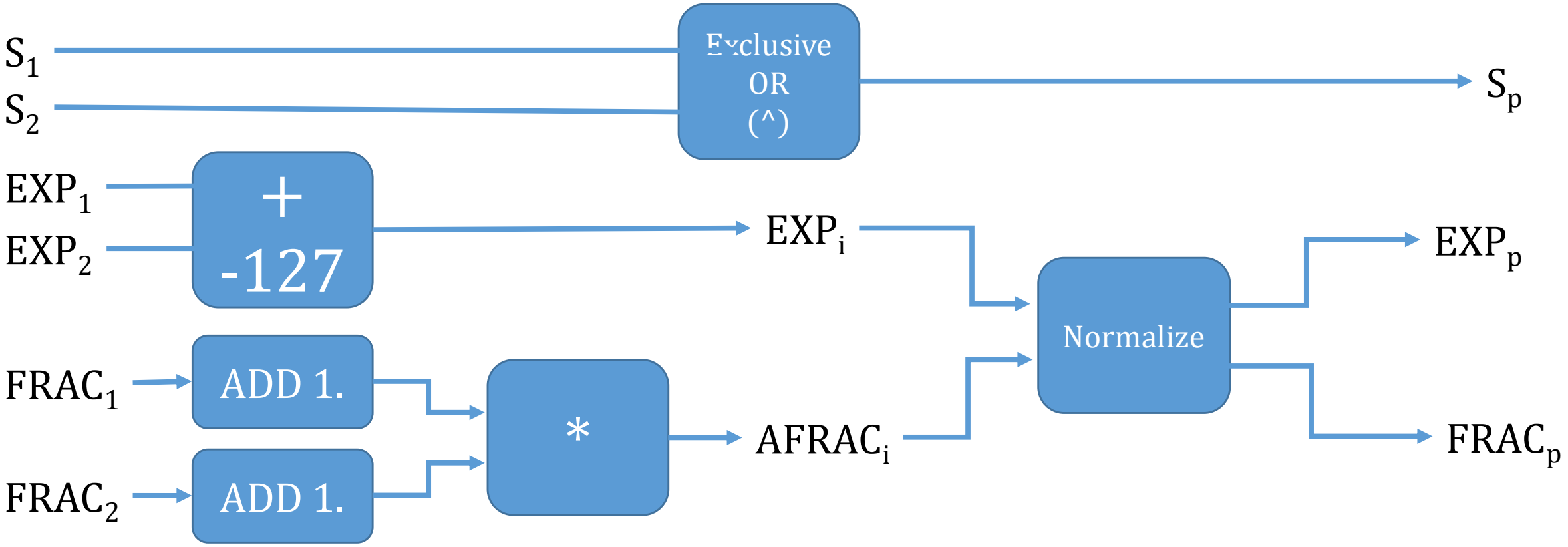
S	EXP								FRAC																					
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Biggest Denormal: $0.11111\ldots. \times 2^{-127} = 1.1754942 \times 10^{-38}$

S	EXP								FRAC																					
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Multiplying two floating point numbers

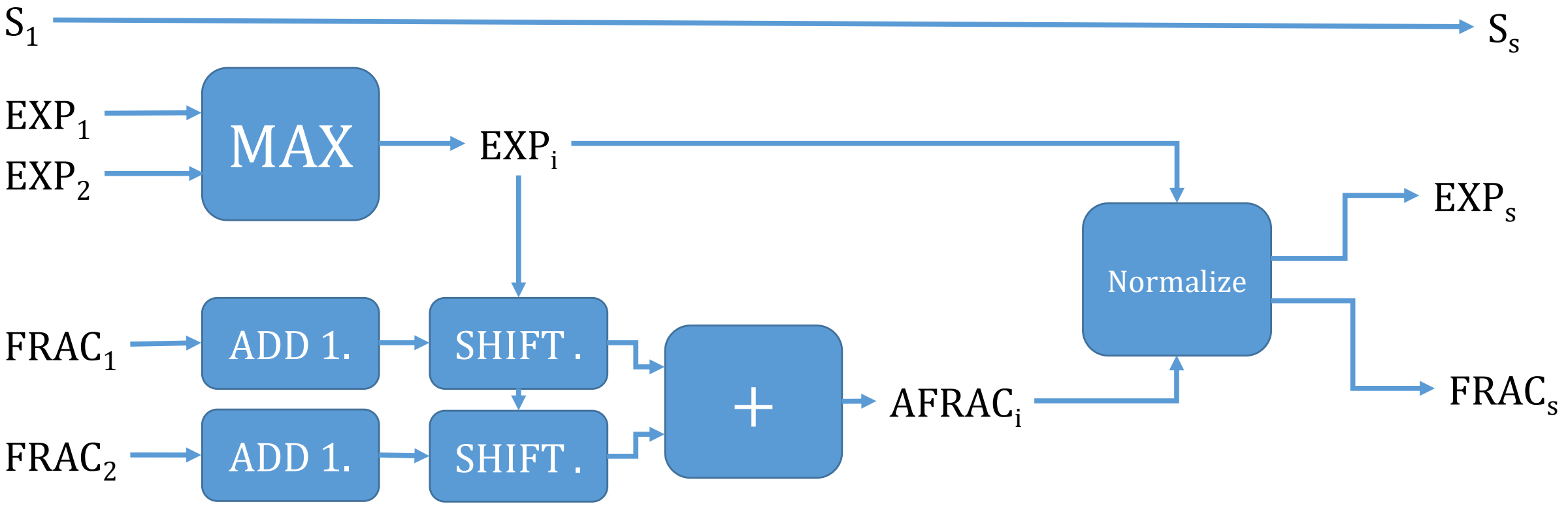
Given S_1, EXP_1, $FRAC_1$; and S_2, EXP_2, $FRAC_2$, Compute S_p, EXP_p, $FRAC_p$



Adding Two Floating Point Numbers

Given S_1, EXP_1, $FRAC_1$; and S_2, EXP_2, $FRAC_2$, Compute S_s, EXP_s, $FRAC_s$

If (S_1 != S_2) { S_1 = !S_1; return SUBTRACT(…)}



Dealing with Precision

BAD

```
float x=1.0/3.0;
float y=1.0 - (2.0/3.0);
if (x==y) {....

}
```

BETTER

```
#define EPSILON 0.0001
float x=1.0/3.0;
float y=1.0 - (2.0/3.0);
if (EPSILON > fabs(x-y)) { ...

}
```

Mixing Integers and Floats

- Converting Float to Int:
  - Express Float as Abstract binary, but truncate after binary point
    int x = 3.1414; // x is 3
    int y = 6.23e22; // y is garbage… doesn't fit in 32 bits
- Converting Int to Float:
  - Add .0 to integer value
    float fx=3; // fx=3.0
  - If the result cannot be represented exactly, round up or down
    float fy=100000001; // fy=1.0 x 10^8

Automatic Conversion

- If anything in an operation is double, operation is evaluated using double precision floating point
- Otherwise, if anything in an operation is float, operation is evaluated using float
- Conversion also occurs to the target type on assignment and argument evaluation

Implicit Casting Gotcha

```
int percent=foo();
int base=bar();

int result = base * (percent/100);
int result = base * (float)(percent/100);
int result = (base * percent)/100;
int result = base * (percent / (float) 100);
int result = base * (percent / 100.0);
```