IEEE 754 Single-Precision Format

- Single-precision numbers contain a total of 32 bits, partitioned as follows:
  - 1-bit sign, S
  - 8-bit biased exponent, E (with a bias of 127), in the range [0, 255].
  - 23-bit fraction, F
- Single-precision normalized numbers have a biased exponent in the range [1, 254].
  - The value of a normalized number is: $(-1)^S(1.F)2^{E-127}$
  - The 24-bit quantity 1.F is the significand, and is in the range [1, 2).
- Single-precision denormalized numbers have a biased exponent of 0 and a non-zero fraction.
  - The value of a denormalized number is: $(-1)^S(0.F)2^{-126}$
  - This allows for "graceful underflow" of very small magnitude numbers.
  - Not all hardware supports denormalized numbers (since it adds complexity and may be slower), and programmers may disable denormalized numbers to increase speed.
- Single-precision zeros have a biased exponent of 0 and a zero fraction.
- Single-precision infinities have a biased exponent of 255 and a zero fraction.
- Single-precision NANs (non-a-number) have a biased exponent of 255 and a non-zero fraction. (The value of the fraction can be used to pass information about an exception.)

IEEE 754 Double-Precision Format

- Double-precision numbers contain a total of 64 bits, partitioned as follows:
  - 1-bit sign, S
  - 11-bit biased exponent, E (with a bias of 1023), in the range [0, 2047].
  - 52-bit fraction, F
- Double-precision normalized numbers have a biased exponent in the range [1, 2046].
  - The value of a normalized number is: $(-1)^S(1.F)2^{E-1023}$
  - The 53-bit quantity 1.F is the significand, and is in the range [1, 2).
- Double-precision denormalized numbers have a biased exponent of 0 and a non-zero fraction.
  - The value of a denormalized number is: $(-1)^S(0.F)2^{-1022}$
- Double-precision zeros have a biased exponent of 0 and a zero fraction.
- Double-precision infinities have a biased exponent of 2047 and a zero fraction.
- Double-precision NANs have a biased exponent of 2047 and a non-zero fraction.

IEEE 754 Rounding Modes

- Round to Nearest Even (RNE):
  - The default rounding mode that must be supported in any implementation.
  - The "nearest even" avoids any rounding bias in the case where the original number is exactly mid-way between two representable numbers.
- Round toward Zero (RZ):
  - Simply truncation.
- Round toward Plus Infinity (RPI):
  - Round in the direction of positive infinity.
- Round toward Minus Infinity (RMI):
  - Round in the direction of negative infinity.
- Note that RPI and RMI are used in *interval arithmetic*, in which a real number x is represented by two floating-point numbers x_1 and x_2 that bracket the number. (x_1 is the closest representable number that is less than or equal to x and x_2 is the closest representable number that is greater than or equal to x). Arithmetic operations are then done on intervals. For example to add real numbers x and y using interval arithmetic:

$$[x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2]$$

where $x_1 + y_1$ is rounded toward -infinity and $x_2 + y_2$ is rounded toward +infinity.

Exceptions

- Overflow:
  - Occurs if the exponent of the result is larger than the maximum allowable value (e.g., for double-precision, if the exponent is larger than 2046). The correct output then depends on the rounding mode and the sign of the result:
    - If RNE: (sign)(infinity)
    - If RZ: (sign)(max. representable number)
    - If RPI: If sign = +, then +infinity; otherwise, -(max. representable number)
    - If RMI: If sign = +, then (max. representable number); otherwise, -infinity
- Underflow:
  - Action depends on whether denormalized numbers are handled or not.
- Divide by 0:
  - If the divisor is 0 and the dividend is non-zero and finite, output = (sign)(infinity)
- Invalid Operation:
  - "weird" operations, such as (0)(infinity), 0/0, (+infinity) + (-infinity), infinity/infinity, sqrt of a negative number, … Calls a trap handler or returns NaN.
- Inexact Result:
  - Sets a flag if the result is not exact.

Basic Sequential Floating-Point Add/Subtract Algorithm

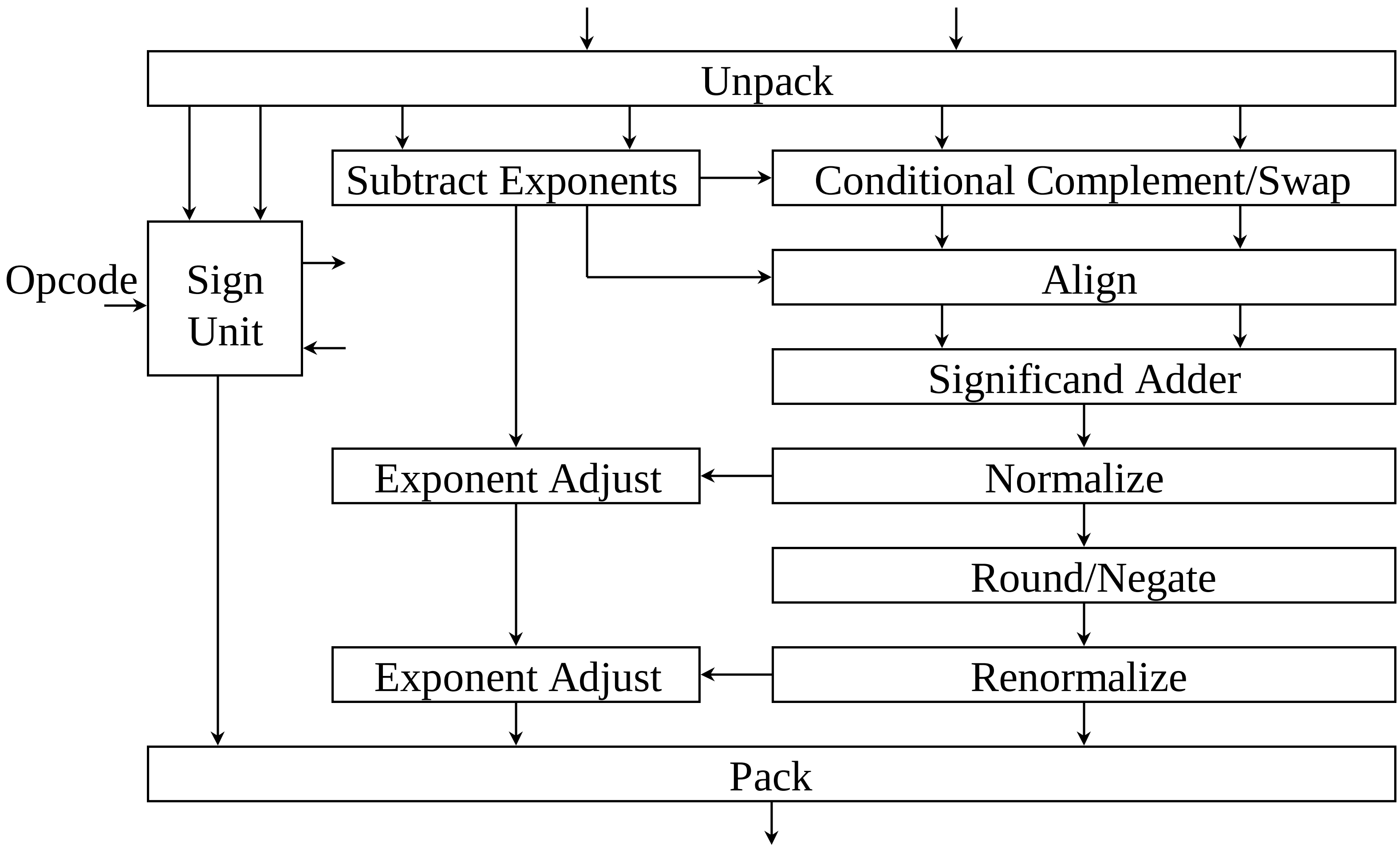
- Given two signed operands and two possible opcodes (i.e., add and subtract):
  - An <u>effective addition</u> will be performed in the following situations:
    - Add opcode with operands having the same sign.
    - Subtract opcode with operands having opposite signs.
  - An <u>effective subtraction</u> will be performed in the following situations:
    - Add opcode with operands having opposite signs.
    - Subtract opcode with operands having the same sign.
- For normalized operands A and B, the basic sequential algorithm for A+B or A-B is:
  - <u>Alignment</u>:
    - If the exponents of the two operands differ, determine which operand has the *smaller* exponent. Right-shift the significand of the smaller operand by an amount E_1 - E_2, where E_1 is the larger exponent and E_2 is the smaller exponent. Change the exponent of the smaller operand to E_1.
  - <u>Add/Subtract</u>:
    - From the signs and the opcode, determine if this is an effective addition or subtraction and perform the corresponding operation on the significands.

Basic Seq. Floating-Point Add/Subtract Algorithm - Cont.

- Normalization:
  - If we performed an effective addition, then the significand of the result will be in the range [1, 4). There are two sub-cases:
    - If it is in the range [1, 2), then no normalization is required.
    - If it is in the range [2, 4), then we must perform a right shift by one bit position, and increment the value of the exponent by one. Overflow may occur.
  - For an effective subtraction (and assuming that the aligned was subtracted from the unaligned), then the result significand be in the range (-1, 2):
    - If it is in the range [1, 2), then no normalization is required.
    - If it is in the range (-1, 1), then we must perform a left shift by one or more bit positions, and decrement the value of the exponent by the amount of the shift. Underflow may occur.
- Rounding/Negation:
  - Round the result and, if necessary, negate the significand and update the sign.
- Renormalization:
  - In rare cases, rounding will produce a result which is no longer normalized. In such cases, another 1-bit shift must be performed (and the exponent adjusted).

Block Diagram for Basic Seq. Floating-Point Add/Subtract

- The basic sequential algorithm can be mapped onto sign, exponent and significand data paths, as shown below. (Note that not all required connections are indicated explicitly.)



Basic Floating-Point Multiply Algorithm

- The three main sub-operations are:
  - Multiply the significands
  - Add the biased exponents and subtract the bias (since each of the two operand exponents contains one copy of the bias)
  - XOR the signs
- We must also perform normalization and rounding, with corresponding adjustments to the biased exponent:
  - If the significand of each operand is normalized, then each one will be in the range [1, 2). Therefore, the product significand will be in the range [1, 4):
    - If the product is in the range [1, 2), then it is already normalized.
    - If the product is in the range [2, 4), then we must shift right by one bit position and increment the exponent by 1.
- For normalized single precision operands, biased exponent are in the range [1, 254], so the sum will be in the range [2, 508]. After subtracting the bias, it is in [-125, 381]:
  - A final exponent < 1 means that the result is too small to be a normalized number.
  - A final exponent > 254 means that the result is too large to be a normalized number.

Subtraction of Bias: IEEE Single-Precision Format

- The bias subtraction does not require a full-width operation because of its special value.
- Note that -127 = 1 - 128, so instead of subtracting 127, we can add 1 and subtract 128.
  - The "add 1" is done as a carry-in to the LSB when the two biased exponents are initially added, creating an unsigned 9-bit number in [3, 509]:

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
+								1
e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

  - The "subtract 128" (where $128 = 2^7$) can be done as follows: Prepend a 0 MSB, creating a 10-bit two's complement number, and add -128 as follows:

	0	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0
+	1	1	1	0	0	0	0	0	0	0
	y	x	$\overline{e_7}$	e_6	e_5	e_4	e_3	e_2	e_1	e_0

  Note that this only requires a 3-bit addition. The sum will be in [-125, 381], which is contained within the range of a 10-bit two's complement number i.e., [-512, 511].
- After any adjustments from normalization and rounding, if x = y = 0 (and the remaining bits are not all 0s or all 1s) the resulting exponent is proper for a normalized number. On the other hand, y = 1 indicates underflow, while y = 0 and x = 1 indicates overflow.

Subtraction of Bias: IEEE Double-Precision Format

- For normalized numbers in double-precision format, the bias is 1023 and the biased exponents are 11-bit unsigned quantities in the range [1, 2046].
- Note that -1023 = 1 - 1024, so we can add 1 and subtract 1024:
  - The "add 1" can be done as a carry-in to the LSB when the two biased exponents are initially added (creating an unsigned 12-bit number in [3, 4093]:

	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
+											1
e_{11}	e_{10}	e_9	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

  - The "subtract 1024" (where 1024= 2^{10}) can be done as follows: Prepend a 0 MSB, creating a 13-bit two's complement number, and add -1024 as follows:

	0	e_{11}	e_{10}	e_9	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0
+	1	1	1	0	0	0	0	0	0	0	0	0	0
	y	x	$\overline{e}_{10}$	e_9	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

- This still only requires a 3-bit addition. The sum is in [-1021, 3069], which is within the range of a 13-bit two's complement number, i.e. [-4096, 4095]. After any adjustments from normalization and rounding, if x = y = 0 (and the remaining bits are not all 0s or all 1s), then the resulting exponent is proper for a normalized number. Otherwise, y = 1 indicates underflow, while y = 0 and x = 1 indicates overflow.

Basic Structure of the Data Path

- The data path is split into sign, exponent and significand data paths as shown below.