Exercise 5: Fixed-Point Arithmetic

1 Problem 1: Fixed-Point Format $(6, -4)$

Problem Statement

Consider the signed fixed-point format $(6, -4)$ (most significant bit position: 6, least significant bit position: -4). Indicate the number of bits in this format, and the smallest and largest values. What is the quantization step?

Solution

Fixed-Point Notation

In the fixed-point format (m, l):

- m = position of the most significant bit (MSB)
- l = position of the least significant bit (LSB)
- The binary point is between bit positions 0 and -1

For a signed fixed-point number in format (m, l), the representation is:

$$x = -b_m \cdot 2^m + \sum_{i=l}^{m-1} b_i \cdot 2^i \tag{1}$$

where $b_i \in \{0, 1\}$ are the bits, and b_m is the sign bit (two's complement).

Number of Bits

The total number of bits is:

$$n = m - l + 1 = 6 - (-4) + 1 = 11 \text{ bits} \tag{2}$$

The bit positions range from 6 (MSB, sign bit) down to -4 (LSB).

Quantization Step

The quantization step (spacing between consecutive representable numbers) is:

$$\Delta = 2^l = 2^{-4} = \frac{1}{16} = 0.0625 \tag{3}$$

Largest Value

The largest value occurs when all bits except the sign bit are 1:

$$x_{\max} = \sum_{i=-4}^{5} 2^i = 2^6 - 2^{-4} \tag{4}$$

$$= 64 - 0.0625 \tag{5}$$

$$= 63.9375 \tag{6}$$

Alternatively:

$$x_{\max} = 2^m - 2^l = 2^6 - 2^{-4} = 63.9375 \tag{7}$$

In binary: $0\,111111\,1111_2$ (sign bit 0, then ten 1's)

Smallest Value

The smallest (most negative) value occurs when only the sign bit is 1:

$$x_{\min} = -2^m = -2^6 = -64 \tag{8}$$

In binary: $1\,000000\,0000_2$ (sign bit 1, then ten 0's)

Summary for Format $(6, -4)$

- **Number of bits**: 11
- **Range**: $[-64, 63.9375]$
- **Quantization step**: $\Delta = 0.0625 = 1/16$
- **Total representable values**: $2^{11} = 2048$

2 Problem 2: Representing Real Numbers in 8-bit Fixed Point

Problem Statement

Represent the following reals in 8-bit signed fixed point, indicating the fixed point format (position of the most significant bit and the least significant bit):

$$9.3452 \qquad 0.03434 \qquad -2.18625$$

Solution

For 8-bit signed fixed point, we have 8 bits total. The format is (m, l) where $m - l + 1 = 8$, so $m = l + 7$.

Representing 9.3452

Step 1: Choose format. The number 9.3452 requires at least $\lceil \log_2(9.3452) \rceil = 4$ bits for the integer part (since $2^3 = 8 < 9.3452 < 16 = 2^4$). With the sign bit, we need bit position at least 3 for the integer magnitude.

Choose format $(3, -4)$ (8 bits: positions 3, 2, 1, 0, -1, -2, -3, -4):

- Bit 3: sign bit and $2^3 = 8$
- Bits 2 down to -4: remaining value
- Range: $[-16, 15.9375]$
- Quantization step: $2^{-4} = 0.0625$

Step 2: Convert to binary. Integer part: $9 = 1001_2$

Fractional part: $0.3452 \times 16 = 5.5232 \Rightarrow$ closest is $0101_2 = 5/16 = 0.3125$

Step 3: Fixed-point representation.

$$9.3452 \approx 9.3125 = 0\,1001\,0101_2 \text{ in format } (3, -4) \tag{9}$$

Binary breakdown:

$$\begin{aligned} &b_3 = 0 \text{ (sign)} \\ &b_2 b_1 b_0 = 100_2 = 8 \\ &b_{-1} b_{-2} b_{-3} b_{-4} = 0101_2 = 5/16 = 0.3125 \\ &\text{Value: } 8 + 1 + 0.3125 = 9.3125 \end{aligned}$$

Error:

- Absolute error: $|9.3452 - 9.3125| = 0.0327$
- Relative error: $0.0327/9.3452 \approx 0.0035 = 0.35\%$

Representing 0.03434

Step 1: Choose format. This is a small fractional number. We need: $2^l \leq 0.03434$

$2^{-5} = 0.03125 < 0.03434 < 0.0625 = 2^{-4}$

Choose format $(-1, -8)$ (8 bits: positions -1, -2, -3, -4, -5, -6, -7, -8):

- Bit -1: sign bit and $2^{-1} = 0.5$
- Range: $[-0.5, 0.4961]$ (approximately)
- Quantization step: $2^{-8} = 0.00390625$

Step 2: Convert to binary. We need to represent 0.03434 as a sum of powers of 2 from 2^{-2} to 2^{-8}:

$$0.03434 \times 2^8 = 8.79$$

$$\text{Closest integer: } 9 = 00001001_2$$

So: $0.03434 \approx 9/256 = 0.03515625$

Step 3: Fixed-point representation.

$$0.03434 \approx 0.03516 = 0\,0001001_2 \text{ in format } (-1, -8) \tag{10}$$

Binary: $0\,0001001_2$
Verification: $0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 0 \cdot 2^{-7} + 1 \cdot 2^{-8} = 2^{-5} + 2^{-8} = 0.03125 + 0.00390625 = 0.03515625$

Error:

- Absolute error: $|0.03434 - 0.03516| \approx 0.00082$
- Relative error: $0.00082/0.03434 \approx 0.024 = 2.4\%$

Representing -2.18625

Step 1: Choose format. The magnitude is approximately 2.19, requiring bits for values up to about 4.
Choose format $(2, -5)$ (8 bits: positions 2, 1, 0, -1, -2, -3, -4, -5):

- Bit 2: sign bit and $2^2 = 4$
- Range: $[-4, 3.96875]$
- Quantization step: $2^{-5} = 0.03125$

Step 2: Convert using two's complement. First, represent positive 2.18625:

$$\begin{aligned} 2.18625 &= 2 + 0.18625 \\ &= 10_2 + 0.18625 \end{aligned}$$

For fractional part: $0.18625 \times 32 = 5.96 \approx 6 = 00110_2$
So $2.18625 \approx 2.1875 = 010\,00110_2$
For negative value (two's complement):

$$\begin{aligned} -2.1875: \quad &\text{Invert: } 101\,11001_2 \\ &\text{Add 1: } 101\,11010_2 \end{aligned}$$

Step 3: Fixed-point representation.

$$-2.18625 \approx -2.1875 = 1\,01\,11010_2 \text{ in format } (2, -5) \tag{11}$$

Verification using two's complement formula:

$$\begin{aligned} &- b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0 + b_{-1} \cdot 2^{-1} + \cdots + b_{-5} \cdot 2^{-5} \\ &= -1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 + 1 \cdot 0.5 + 1 \cdot 0.25 + 0 \cdot 0.125 + 1 \cdot 0.0625 + 0 \cdot 0.03125 \\ &= -4 + 1 + 0.5 + 0.25 + 0.0625 \\ &= -2.1875 \end{aligned}$$

Error:

- Absolute error: $|-2.18625-(-2.1875)| = 0.00125$
- Relative error: $0.00125/2.18625 \approx 0.00057 = 0.057\%$

Summary Table

Value	Format	Binary	Approximation	Abs. Error	Rel. Error
9.3452	$(3,-4)$	01001010_2	9.3125	0.0327	0.35%
0.03434	$(-1,-8)$	00001001_2	0.03516	0.00082	2.4%
-2.18625	$(2,-5)$	10111010_2	-2.1875	0.00125	0.057%

3 Problem 3: Error Bounds in Fixed-Point Arithmetic

Problem Statement

Indicate the relative and absolute error for each representation in Problem 2. In a general way, how to bound these two errors as a function of the width w used to represent the number?

Solution

Errors from Problem 2

See the summary table above for specific errors.

General Error Bounds

For a fixed-point format (m, l) with $w = m - l + 1$ bits (including sign bit):

Quantization step:

$$\Delta = 2^l \tag{12}$$

Absolute error bound: When rounding to nearest:

$$|E_{\text{abs}}| \leq \frac{\Delta}{2} = \frac{2^l}{2} = 2^{l-1} \tag{13}$$

When truncating (round toward zero):

$$|E_{\text{abs}}| \leq \Delta = 2^l \tag{14}$$

Relationship between w, m, and l: Given width w and choosing format (m, l):

$$l = m - w + 1 \tag{15}$$

Therefore:

$$\Delta = 2^l = 2^{m-w+1} = \frac{2^{m+1}}{2^w} \tag{16}$$

Absolute error bound in terms of w: For round-to-nearest:

$$|E_{\text{abs}}| \leq \frac{2^{m+1}}{2^{w+1}} = \frac{2^m}{2^w} \tag{17}$$

Key insight: The absolute error decreases exponentially with the bit width w. Each additional bit cuts the error in half.

Relative error bound: For a value $x \neq 0$ in range $[2^{m-1}, 2^m)$:

$$|E_{\text{rel}}| = \frac{|E_{\text{abs}}|}{|x|} \leq \frac{2^{l-1}}{2^{m-1}} = 2^{l-m} = 2^{-(m-l)} = 2^{-(w-1)} \tag{18}$$

For round-to-nearest with x near maximum magnitude:

$$|E_{\text{rel}}| \leq \frac{1}{2^{w-1}} \approx \frac{1}{2^w} \tag{19}$$

General bounds:

$$|E_{\text{abs}}| \leq 2^{l-1} = \frac{\Delta}{2} \tag{20}$$

$$|E_{\text{rel}}| \leq 2^{-(w-1)} = \frac{1}{2^{w-1}} \tag{21}$$

Interpretation:

- The absolute error depends on where the binary point is placed (determined by l)
- The relative error depends on the total number of bits w (precision)
- Each additional bit improves relative accuracy by a factor of 2
- For w-bit fixed point: roughly $w-1$ bits of precision (sign bit doesn't contribute to precision)

Comparison with Problem 2 Results

For $w = 8$ bits, the theoretical relative error bound is:

$$|E_{\text{rel}}| \leq 2^{-7} = \frac{1}{128} \approx 0.78\% \tag{22}$$

Our actual results:

- 9.3452: 0.35% (below bound)
- 0.03434: 2.4% (exceeds bound)
- -2.18625: 0.057% (well below bound)

The case of 0.03434 exceeds the bound because the number is small relative to the format's range. The relative error bound assumes the number is reasonably close to the maximum representable magnitude. For numbers much smaller than 2^m, relative error can be larger.

4 Problem 4: Fixed-Point Computation of Linear Combination

Problem Statement

We want to calculate:

$$9.3452x + 0.03434y - 2.18625z \tag{23}$$

where x, y, and z are variables known to belong to the intervals $[-24; 3.4]$, $[-150; 2500]$, and $[-112; 18]$ respectively. Propose a fixed-point format for these three variables, as well as a fixed-point realization (set of mantissa operations) to best perform this operation, assuming we have an 8-bit by 8-bit multiplier giving a 16-bit result, and a 16-bit accumulator (plus 4 guard bits).

Solution

Step 1: Choose Fixed-Point Formats for Variables

Variable $x \in [-24, 3.4]$**:** Maximum magnitude: $24 < 32 = 2^5$

Choose format $(\mathbf{4}, \mathbf{-3})$ for x:

- Range: $[-16, 15.875]$ — insufficient for -24!

Choose format $(\mathbf{5}, \mathbf{-2})$ for x:

- 8 bits: positions 5, 4, 3, 2, 1, 0, -1, -2
- Range: $[-32, 31.75]$
- Quantization: $\Delta = 2^{-2} = 0.25$

Variable $y \in [-150, 2500]$**:** Maximum magnitude: $2500 < 4096 = 2^{12}$

Choose format $(\mathbf{11}, \mathbf{4})$ for y:

- 8 bits: positions 11, 10, 9, 8, 7, 6, 5, 4
- Range: $[-2048, 2032]$ — insufficient for 2500!

Choose format $(\mathbf{12}, \mathbf{5})$ for y:

- 8 bits: positions 12, 11, 10, 9, 8, 7, 6, 5
- Range: $[-4096, 4064]$
- Quantization: $\Delta = 2^5 = 32$

Variable $z \in [-112, 18]$**:** Maximum magnitude: $112 < 128 = 2^7$

Choose format $(\mathbf{6}, \mathbf{-1})$ for z:

- 8 bits: positions 6, 5, 4, 3, 2, 1, 0, -1
- Range: $[-64, 63.5]$ — insufficient for -112!

Choose format $(\mathbf{7}, \mathbf{0})$ for z:

- 8 bits: positions 7, 6, 5, 4, 3, 2, 1, 0
- Range: $[-128, 127]$
- Quantization: $\Delta = 2^0 = 1$

Step 2: Choose Fixed-Point Formats for Coefficients

From Problem 2, we established:

- $c_1 = 9.3452$ in format $(3, -4)$
- $c_2 = 0.03434$ in format $(-1, -8)$
- $c_3 = -2.18625$ in format $(2, -5)$

Step 3: Analyze Multiplication Results

When multiplying two fixed-point numbers in formats (m_1, l_1) and (m_2, l_2), the product is in format $(m_1 + m_2, l_1 + l_2)$ (before considering sign bit growth).

Product 1: $c_1 \cdot x$

$$\begin{aligned} c_1 &: (3, -4) \text{ format, 8 bits} \\ x &: (5, -2) \text{ format, 8 bits} \\ c_1 \cdot x &: (3 + 5, -4 + (-2)) = (8, -6) \text{ format, 16 bits} \end{aligned}$$

Range: Maximum value $\approx 9.3452 \times 24 \approx 224$, which is $< 2^8 = 256$

Product 2: $c_2 \cdot y$

$$\begin{aligned} c_2 &: (-1, -8) \text{ format, 8 bits} \\ y &: (12, 5) \text{ format, 8 bits} \\ c_2 \cdot y &: (-1 + 12, -8 + 5) = (11, -3) \text{ format, 16 bits} \end{aligned}$$

Range: Maximum value $\approx 0.03434 \times 2500 \approx 85.9$, which is $< 2^{11} = 2048$

Product 3: $c_3 \cdot z$

$$\begin{aligned} c_3 &: (2, -5) \text{ format, 8 bits} \\ z &: (7, 0) \text{ format, 8 bits} \\ c_3 \cdot z &: (2 + 7, -5 + 0) = (9, -5) \text{ format, 16 bits} \end{aligned}$$

Range: Maximum value $\approx 2.18625 \times 112 \approx 245$, which is $< 2^9 = 512$

Step 4: Align Products for Accumulation

The three products have different LSB positions. For accumulation, we need to align them. The accumulator is 16 bits with 4 guard bits, giving 20 total bits.

Analysis of product formats:

- Product 1: $(8, -6)$ — LSB at position -6
- Product 2: $(11, -3)$ — LSB at position -3
- Product 3: $(9, -5)$ — LSB at position -5

To accumulate, we should align to the finest resolution, which is position -6 (from Product 1).

Alignment strategy: Choose accumulator format $(\mathbf{13}, \mathbf{-6})$ (20 bits including 4 guard bits):

- 20 bits: positions 13 down to -6
- Range: $[-8192, 8191.984375]$
- This can hold the maximum possible sum: $224 + 85.9 + 245 \approx 555$

Step 5: Scaling Operations

To align products to the accumulator format:

Product 1: $(8, -6) \to (13, -6)$ No scaling needed, just sign-extend to 20 bits (add 4 leading bits matching the sign bit).

Product 2: $(11, -3) \to (13, -6)$ Shift left by 3 bits (multiply by $2^3 = 8$) to move LSB from position -3 to -6. Add 2 leading sign bits to reach position 13.

Product 3: $(9, -5) \to (13, -6)$ Shift left by 1 bit (multiply by $2^1 = 2$) to move LSB from position -5 to -6. Add 4 leading sign bits to reach position 13.

Step 6: Computation Sequence

1. **Multiply**: $P_1 = c_1 \times x$ (8-bit $\times$ 8-bit $\to$ 16-bit result)
2. **Sign-extend**: Extend P_1 to 20 bits $\to P_1'$
3. **Initialize accumulator**: $A = P_1'$
4. **Multiply**: $P_2 = c_2 \times y$ (8-bit $\times$ 8-bit $\to$ 16-bit result)
5. **Shift and extend**: Shift P_2 left by 3 bits, extend to 20 bits $\to P_2'$
6. **Accumulate**: $A = A + P_2'$
7. **Multiply**: $P_3 = c_3 \times z$ (8-bit $\times$ 8-bit $\to$ 16-bit result)
8. **Shift and extend**: Shift P_3 left by 1 bit, extend to 20 bits $\to P_3'$
9. **Accumulate**: $A = A - P_3'$ (note: subtraction since c_3 is negative is handled by sign)
10. **Result**: A in format $(13, -6)$ contains the final result

Summary of Formats

Alternative: Normalization Approach

An alternative approach is to scale all variables to have similar dynamic ranges before computation:

1. Scale y by $1/32$: $y' = y \cdot 2^{-5}$, so $y' \in [-4.69, 78.125] \to$ format $(7, 0)$
2. Adjust coefficient: $c_2' = c_2 \cdot 32 = 1.0989 \to$ format $(0, -7)$
3. This makes all products have comparable magnitudes and simplifies alignment

However, this approach requires preprocessing and may not be suitable for all applications.

Variable/Coefficient	Format	Bits
x	$(5, -2)$	8
y	$(12, 5)$	8
z	$(7, 0)$	8
$c_1 = 9.3452$	$(3, -4)$	8
$c_2 = 0.03434$	$(-1, -8)$	8
$c_3 = -2.18625$	$(2, -5)$	8
Product $c_1 \cdot x$	$(8, -6)$	16
Product $c_2 \cdot y$	$(11, -3)$	16
Product $c_3 \cdot z$	$(9, -5)$	16
Accumulator	$(13, -6)$	20 (16+4 guard)

Key Considerations

- **Overflow prevention**: The accumulator format must accommodate the maximum possible sum/difference
- **Precision preservation**: Use the finest quantization step (here 2^{-6}) for the accumulator
- **Guard bits**: The 4 guard bits prevent overflow during intermediate additions
- **Alignment cost**: Shifting operations have minimal hardware cost but must be carefully tracked
- **Rounding**: After final accumulation, result may need rounding if converting to narrower format