# DSP Number Systems
# Fixed Point Numbers

# Today's Key Points

- Arduino Data Types
- Number Representations
  - Fixed Point
  - Floating Point

- Working with fixed point numbers systems
  - 2's complement
  - Adding/subtracting numbers
  - Number wrapping

- QM.N number representations

# DSP Data Representation

- Many different types of information can be represented in our DSP system

- Each are stored as bit patterns within the processor

# Arduino Data Types

| Data Type | Used For | Range | Number of Bytes | Number of Bits | Notes |
|---|---|---|---|---|---|
| Boolean | Holding True or False values | true or false | 1 | 8 | |
| char | Holds an ASCII character value | A, B, ..0,1,2,..$,#, etc.. | 1 | 8 | |
| byte | Stores an unsigned number | 0-255 | 1 | 8 | |
| int | Stores a signed integer value | -32768 to 32767 | 2 | 16 | 32 bits on a Due |
| unsigned int | Stores an unsigned integer | 0 to 65535 | 2 | 16 | 32 bits on a Due |
| word | Stores an unsigned integer | 0 to 65535 | 2 | 16 | |
| long | Stores a signed integer number | -2147483648 to 2147483647 | 4 | 32 | |
| unsigned long | Stores a signed integer number | 0 to 4294967295 | 4 | 32 | |
| short | Stores a signed integer number | -32768 to 32767 | 2 | 16 | |
| float | Stores a signed floating point number | -3.4028e38 to 3.4028e38 | 4 | 32 | On UNO same as a float |
| double | Stores a signed floating point number | -3.4028e38 to 3.4028e38 | 4 | 32 | |

# ASCII Character Set

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com



http://www.asciitable.com/

# DSP Number Representations

- Numbers in a DSP are represented by bit patterns.
- The interpretation of the bit patterns is the critical part.
- Two main types of numbers are:
  - Fixed Point
  - Floating Point numbers

# Fixed Point Number Representation

- The bit patterns that are stored can represent different numerical values
  - Unsigned Integer (only represent positive numbers)
  - Two’s complement  (closest to actual hardware)
  - Sign and magnitude (easy for humans to understand)
  - Offset binary (used for ADC and DAC)

# Unsigned Integers 4 Bit Example

- Represents positive numbers from $0 \; to \; 2^n - 1$
- For 4-bits 0 to 15

| Unsigned Integers -- 4 Bits | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 15 | 1111 |
| 14 | 1110 |
| 13 | 1101 |
| 12 | 1100 |
| 11 | 1011 |
| 10 | 1010 |
| 9 | 1001 |
| 8 | 1000 |
| 7 | 0111 |
| 6 | 0110 |
| 5 | 0101 |
| 4 | 0100 |
| 3 | 0011 |
| 2 | 0010 |
| 1 | 0001 |
| 0 | 0000 |

# Unsigned Integers – 16 Bit Values

- In Arduino C code we have an UNSIGNED INT that is16-bits in length
- Values from 0 to 65535

| Unsigned Integers -- 16-Bits | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 65535 | 1111111111111111 |
| 65534 | 1111111111111110 |
| 65533 | 1111111111111101 |
| … | … |
| 2 | 0000000000000010 |
| 1 | 0000000000000001 |
| 0 | 0000000000000000 |

# Offset Binary Integers
# 4 Bit Example

- Represents positive numbers from $-2^{\mathrm{n}-1}$ *to* $2^{n-1}-1$
- For 4-bits -8 to 7
- The format is non-standardized
  - Sometimes from -8 to 7
  - Sometimes from -7 to 8

| Offset Binary | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 7 | 1111 |
| 6 | 1110 |
| 5 | 1101 |
| 4 | 1100 |
| 3 | 1011 |
| 2 | 1010 |
| 1 | 1001 |
| 0 | 1000 |
| -1 | 0111 |
| -2 | 0110 |
| -3 | 0101 |
| -4 | 0100 |
| -5 | 0011 |
| -6 | 0010 |
| -7 | 0001 |
| -8 | 0000 |

# Two's Complement 4-Bit Example

- Most "hardware friendly" representation of integer values

Sign Bit Value

| 0 | 010 |
|---|---|

$= +2$

Sign Bit Value

| 1 | 011 |
|---|---|

$= -5$

| Two's Complement -- 4-Bits | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 7 | 0111 |
| 6 | 0110 |
| 5 | 0101 |
| 4 | 0100 |
| 3 | 0011 |
| 2 | 0010 |
| 1 | 0001 |
| 0 | 0000 |
| -1 | 1111 |
| -2 | 1110 |
| -3 | 1101 |
| -4 | 1100 |
| -5 | 1011 |
| -6 | 1010 |
| -7 | 1001 |
| -8 | 1000 |

# Two's Complement
# 16-Bit Example

- 16-bit range from -32768 to +32767

Sign Bit | Value

| 0 | 101000000010101 |
|---|---|

$= +20501$

Sign Bit | Value

| 1 | 101011111110000 |
|---|---|

$= -10256$

| Two's Complement -- 16 Bits | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 32767 | 0111111111111111 |
| 32766 | 0111111111111110 |
| 32765 | 0111111111111101 |
| … | … |
| 2 | 0000000000000010 |
| 1 | 0000000000000001 |
| 0 | 0000000000000000 |
| -1 | 1111111111111111 |
| -2 | 1111111111111110 |
| -3 | 1111111111111101 |
| … | … |
| -32766 | 1000000000000010 |
| -32767 | 1000000000000001 |
| -32768 | 1000000000000000 |

# ADC Output Values

- ADC's often offer two output options
- Straight Binary
- Two's Complement Output
- The range of the output values depends on the choice
- Application dependent

| Unsigned Integers -- 4 Bits | | | Two's Complement -- 4-Bits | |
|---|---|---|---|---|
| Decimal | Bit Pattern | | Decimal | Bit Pattern |
| 15 | 1111 | 5V | 7 | 0111 |
| 14 | 1110 | | 6 | 0110 |
| 13 | 1101 | | 5 | 0101 |
| 12 | 1100 | | 4 | 0100 |
| 11 | 1011 | | 3 | 0011 |
| 10 | 1010 | | 2 | 0010 |
| 9 | 1001 | | 1 | 0001 |
| 8 | 1000 | 2.5V | 0 | 0000 |
| 7 | 0111 | | -1 | 1111 |
| 6 | 0110 | | -2 | 1110 |
| 5 | 0101 | | -3 | 1101 |
| 4 | 0100 | | -4 | 1100 |
| 3 | 0011 | | -5 | 1011 |
| 2 | 0010 | | -6 | 1010 |
| 1 | 0001 | | -7 | 1001 |
| 0 | 0000 | 0V | -8 | 1000 |

# Finding the 2's Complement Positive Number Example

- Start with the number and "complement" each bit
- Add one to the number

Sign Bit | Value

| | Sign Bit | Value | |
|---|---|---|---|
| | 0 | 101000000010101 | $= +20501$ |
| Invert each bit | 1 | 010111111101010 | |
| Add one | 1 | 010111111101011 | $= -20501$ |

# Finding the 2's Complement Negative Number Example

- Start with the number and "complement" each bit
- Add one to the number

| | Sign Bit | Value | |
|---|---|---|---|
| | 1 | 111001001011111 | $= -3489$ |
| Invert each bit | 0 | 000110110100000 | |
| Add one | 0 | 000110110100001 | $= +3489$ |

# 2's Complement Math

- Adding 2's complement numbers is done simply with binary addition
  - 0+0 = 0, 0+1 = 1, 1+1 = 0 with a carry
- Add 3,489 + 23,732

| | | |
|---|---|---|
| 0 | 000110110100001 | $= +3{,}489$ |

$+$

| | | |
|---|---|---|
| 0 | 101110010110100 | $= +23{,}732$ |

$=$

| | | |
|---|---|---|
| 0 | 110101001010101 | $= +27{,}221$ |

# 2's Complement Math
# Subtraction

- Subtracting two positive numbers is done by adding the two's complement
- Subtract 4634 from 10678 $C = 10678 - 4634$

| | | | |
|---|---|---|---|
| | 0 | 010100110110110 | $= +10{,}678$ |
| | | + | |
| 2's complement of +4634 | 1 | 110110111100110 | $= -4{,}634$ |
| | | = | |
| | 0 | 001011110011100 | $= +6{,}044$ |

# What Happens If I Add 2 Large Positive Numbers?

- Add 20,468 to 15,345

| | | |
|---|---|---|
| 0 | 100111111110100 | $= +20{,}468$ |

$+$

| | | |
|---|---|---|
| 0 | 011101111110001 | $= +15{,}345$ |

$=$

| | | |
|---|---|---|
| 1 | 000101111100101 | $= -29{,}723$ |

-29,723 --- Why?

# Wrapping of Integer Values

- If I add two numbers and the sum is larger than the maximum value, the value will “wrap” around.

- Example 6+3 = -7

6+1 = 7
7+1 = -8
-8+1 = -7

| Two's Complement -- 4-Bits | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 7 | 0111 |
| 6 | 0110 |
| 5 | 0101 |
| 4 | 0100 |
| 3 | 0011 |
| 2 | 0010 |
| 1 | 0001 |
| 0 | 0000 |
| -1 | 1111 |
| -2 | 1110 |
| -3 | 1101 |
| -4 | 1100 |
| -5 | 1011 |
| -6 | 1010 |
| -7 | 1001 |
| -8 | 1000 |

+1

+1

+1

# Wrapping of Integer Values

- If I subtract two numbers and the result is smaller than the minimum value, the value will “wrap” around.

- Example -6 - 4 = +6

$$-6 - 1 = -7$$
$$-7 - 1 = -8$$
$$-8 - 1 = 7$$
$$7 - 1 = 6$$

| Two's Complement -- 4-Bits | |
|---|---|
| **Decimal** | **Bit Pattern** |
| 7 | 0111 |
| 6 | 0110 |
| 5 | 0101 |
| 4 | 0100 |
| 3 | 0011 |
| 2 | 0010 |
| 1 | 0001 |
| 0 | 0000 |
| -1 | 1111 |
| -2 | 1110 |
| -3 | 1101 |
| -4 | 1100 |
| -5 | 1011 |
| -6 | 1010 |
| -7 | 1001 |
| -8 | 1000 |

# Fixed Point Numbers
# In Class Problem

- Find the following values assuming an 8-bit number system
- 2's complement of
  - $106d = 01101010b$
  - $-45d = 11010011b$

- Add 79d (01001111b) and 35d (00100011b)
- Subtract 118d (01110110b) from 95d (01011111b)
- Add 85d (01010101b) and 106d (01101010b)

# Fixed Point Numbers
# In Class Problem

- 2's complement of $106d = 01101010b$

Invert $01101010b$ ➡ $10010101b + 00000001b = 10010110b$
$= -106d$

- $-45d = 11010011b$

Invert $11010011$ ➡ $00101100b + 00000001b = 00101101b$
$= 45d$

# Fixed Point Numbers In Class Problem

- Add 79d (01001111b) and 35d (00100011b)

$$\begin{array}{rl} & 01001111b \\ + & 00100011b \\ \hline & 01110010b \;=\; 114d \end{array}$$

# Fixed Point Numbers
# In Class Problem

- Subtract 118d (01110110b) from 95d (01011111b)

Take the 2's complement of 118d

Invert 01110110b ⇨ $10001001b + 00000001b = 10001010b$
$= -118d$

Add the values

01011111b
+ 10001010b

11101001b = -23d

Final result

Invert 11101001b ⇨ $00010110b + 00000001b = 00010111b$

2's complement $= 23d$

# Fixed Point Numbers
# In Class Problem

- Add 85d (01010101b) and 106d (01101010b)

$$\begin{array}{r} 01010101b \\ +\quad 01101010b \\ \hline 10111111b \end{array} = -65d$$

Final result

It's negative. Figure out what is its 2's complement value

Invert 10111111b $\Rightarrow$ $01000000b + 00000001b = 01000001b$

2's complement $= 65d$

# Integers as Counters

- Be careful when using an integer (INT) as a counter value
- The value can “rollover” if the counter exceeds 32767 and will become negative
- If necessary, use a larger fixed point datatype in your code (e.g. a LONG), or add a software check

# Arduino Fixed Point Number Data Types

- Fixed point numbers are primarily stored in the Arduino UNO as INT or LONG data types

- INT is stored as a 16-bit value and requires 2 bytes of storage

- LONG is stored as a 32-bit value and requires 4 bytes of storage

# Fixed Point Number Representation

- Fixed point values represent their value exactly
- When adding or subtracting two fixed point numbers as integers, there is no error
- An integer value will always be the result
- Fixed point numbers have the same increment between values across the entire range of numbers

# Representing Fractions with Fixed Point Values

- Start with a fractional number and convert to a fixed point value
  - Scale the number to preserve the fractional portion
  - It can be scaled by as large a number as desired within the range of the fixed point number system

# Representing Fractions with Fixed Point Values

- Scale by different values to preserve resolution

$$0.4589 \times 10 = 4.589$$

0000000000000100

$$4/10 = 0.4$$

$$0.4589 \times 1000 = 458.9$$

0000000011100101

$$458d/1000 = .458$$

$$0.4589 \times 100000 = 45890.0$$

1011001101000010

Will not fit into SIGNED INT, but will fit into UNSIGNED INT

# Representing Fractions with Fixed Point Values

- Must keep track of the scalar throughout processing
  - Understand the true value of the number
  - Can only add numbers with the same scalar value

$$0.4589 \times 1000 = 458.9$$

0000000011100101

$$458d/1000 = .458$$

$$0.0634 \times 10000 = 634$$

0000001001111010

$$634d/10000 = .0634$$

# Representing Fractions with Fixed Point Values

- Cannot add fixed point values that represent fractions if the scalar is not the same

| | | |
|---|---|---|
| | 0000000011100101 | $458 \rightarrow .4589$ |
| $+$ | 0000001001111010 | $634 \rightarrow .0634$ |
| | 0000010001000100 | $1092 \rightarrow \neq .5223$ |

Cannot add numbers that use different scale values

# Fixed Point Numbers Can Also Represent Fractions

- Treat the MSB as a sign bit
- The remaining 15 bits represent a fraction $< \underline{+} 1.0$

Fractional value

Sign Bit

| 0 | 101000000010101 |
|---|---|

$$= +\frac{20501}{32768} = 0.6257$$

Binary Point Location

Sign Bit

$\boxed{0.101000000010101}$

# Fixed Point Numbers Can Also Represent Fractions

- This approach uses a scalar value, but the scalar is a power of 2
- Example: Represent +0.6257

20501 represents the fractional value

Sign Bit

$$0.6257 \times 2^{15} = 20501$$

Scalar

| 0 | 101000000010101 |
|---|---|

Binary Point Location

Sign Bit

| 0.101000000010101 |
|---|

# Fixed Point Numbers Can Also Represent Fractions

- Negative values have a 1 for the sign bit
- Scale the fraction by $2^{15} = 32768$ for values $<\pm 1.0$
- Use 2's complement to represent negative values

$$-0.3265 \times 2^{15} = -10699$$

Sign Bit

Value

| Sign Bit | Value |
|---|---|
| 1 | 101011000110101 |

Binary Point Location

Sign Bit

| |
|---|
| 1.101011000110101 |

10699 = 2's complement of the binary value

# Different Ranges of Fractions can be represented

- Sign Bit located in the usual position
- Binary Point moved to adjust range
- Different scalar value used (e.g. $2^{13}$)
- Can represent values < $\sim \pm 4.0$

Binary Point Location

$$0\underbrace{10}_{\substack{\text{2 bits} \\ 2}}.\underbrace{1000000010101}_{\substack{\text{13 bits} \\ 4117}}$$

Sign Bit

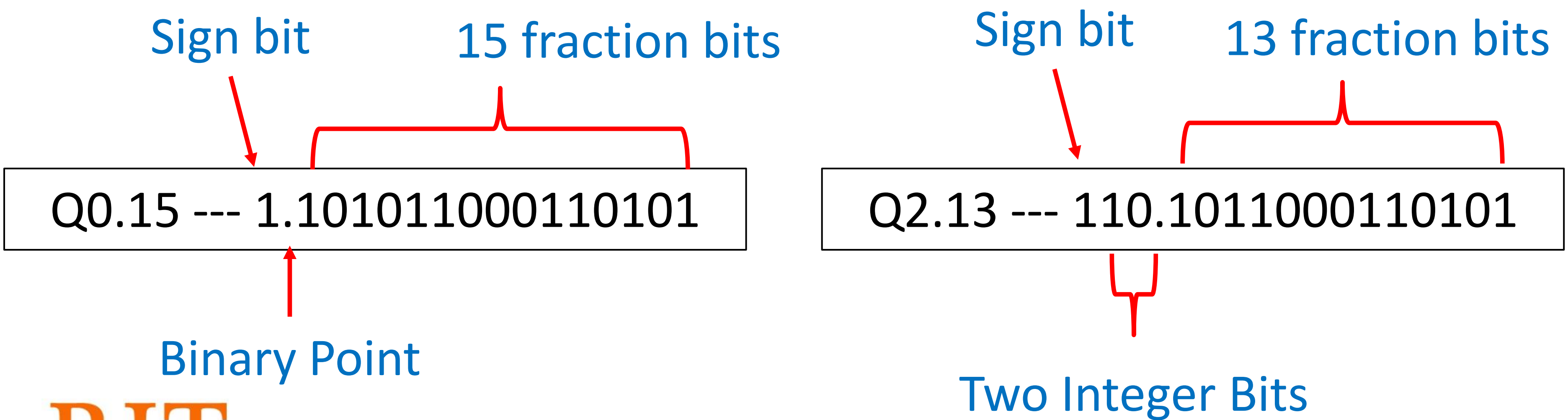
$$\text{Value} = +2 + \frac{4117}{2^{13}} = 2.5026$$

# Using Integers to Represent Fractions

- This is commonly used in hardware (e.g. FPGA's to represent fractions) and in software
- It is not a different data type but a way to interpret the values
- To add or subtract values the binary points must be aligned then the operation performed

# Using Integers to Represent Fractions

- The representation is sometimes written as QM.N (this definition varies)
  - M is the number of bits in the integer portion
  - N is the number of bits in the fraction
  - 1 sign bit is assumed



# Adding Fractional Values

- Adding and subtracting must be done carefully
- Software libraries often written to support the math

Binary Point Location

Sign Bit

| | | |
|---|---|---|
| | 1.101011000110101 | $-0.3265$ |
| $+$ | 010.1000000010101 | $2.5026$ |
| | $\boxed{0010.001011010001}$001 | $2.1761$ |

Truncate to 16 bits

Note new Binary Point Location

# ICP Representing Fractions

- Represent the following numbers as fixed point values using Q0.15 values
  - .2458
  - .06589
  - .000158

# ICP Representing Fractions

- Represent the following numbers as fixed point values using Q0.15 values

$$.2458 \times 32768 = 8054.1286$$

8054

$\boxed{0.001111101110110}$

Sign Bit

Binary Point Location

# ICP Representing Fractions

- Represent the following numbers as fixed point values using Q0.15 values

$$.06589 \times 32768 = 2159.0176$$

2159

0.000100001101111

Sign Bit

Binary Point Location

# ICP Representing Fractions

- Represent the following numbers as fixed point values using Q0.15 values

$$.00158 \times 32768 = 5.177$$

5

$\boxed{0.000000000101}$

Sign Bit

Binary Point Location