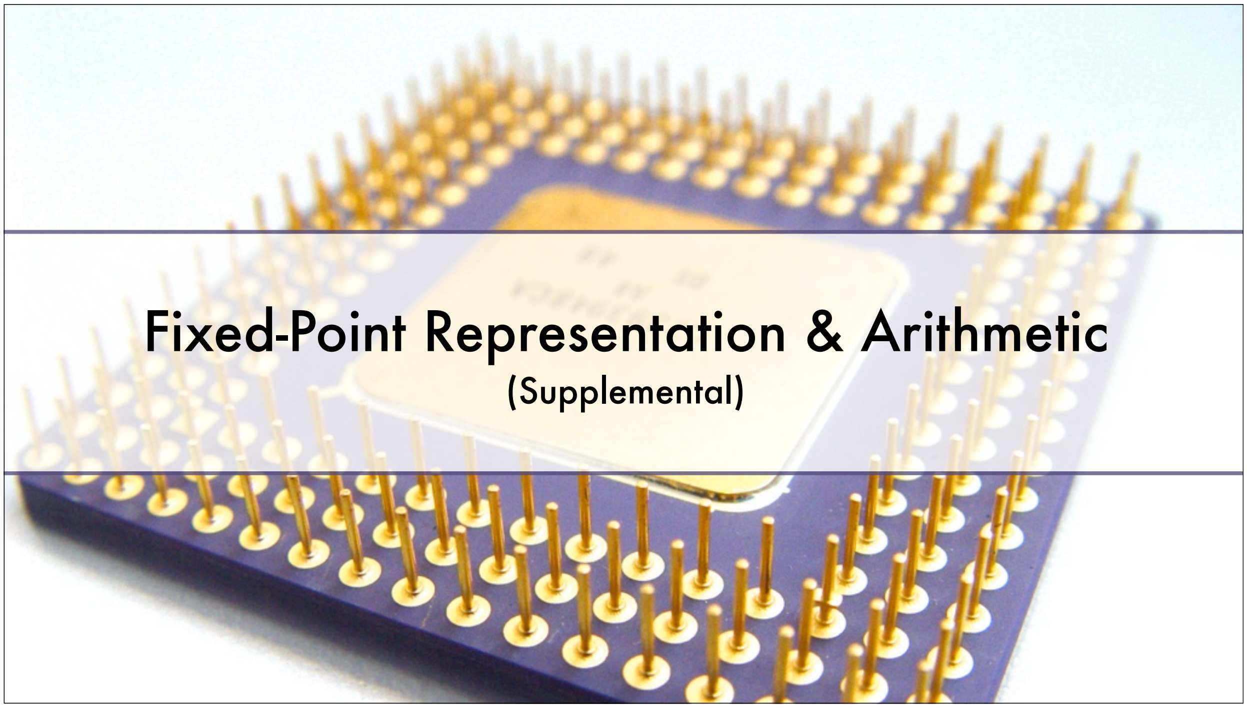


Fixed-Point Representation & Arithmetic

(Supplemental)

– Part 1: Concepts & Representation –

Administrivia

- **Exam 2** Wednesday, November 5
  - Make-up exams must be scheduled **before** the exam is given in class; no make-ups afterward
- **Study Guide** will be posted today/tonight
- HW5 solutions will be posted Sunday after the 48-hour late window closes
- **Lab** Monday (2119/2122 Shelby)
  - Ask questions about exam and homework then

Place Values & Radix Point

- Place values for decimal numbers with a fractional part:

	2	**0**	**1**	**4**	**.**	**5**	**1**	**0**	
...	10^3	10^2	10^1	10^0	**.**	10^{-1}	10^{-2}	10^{-3}	...
...	1000	100	10	1	**.**	1/10	1/100	1/1000	...

- Place values for binary numbers with a fractional part:

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

  - So $1011.0011_2 = 2^3 + 2^1 + 2^0 + 2^{-3} + 2^{-4} = 8 + 2 + 1 + 1/8 + 1/16 = 11^3/_{16}$
- Terminology: the "**.**" is called a
  - decimal point when you're writing a decimal number (2014**.**51)
  - **binary point** when you're writing a binary number (1011.0011_2)
  - **radix point** in general

Activity 16 #1

Converting Unsigned Decimal to Binary

- Integer part: (we did this back in Lecture 2)
  - Divide by 2 until quotient is 0
  - Write remainders in reverse
- Fractional part:
  - Multiply fractional part by 2 (ignoring integer part)
    until product is 0 or sufficient digits have been obtained
  - Write integer parts in order

Activity 16 #2

Finite Binary Representation

- Recall that 1/3 does not have a finite decimal representation (1/3 = 0.3333333333...)
- Some numbers do not have a finite binary representation
  - Example: $1/10 = 0.0\underline{0011}00110011\ldots_2$

    .1 × 2 = **0**.2
    .2 × 2 = **0**.4
    .4 × 2 = **0**.8
    .8 × 2 = **1**.6
    .6 × 2 = **1**.2
    .2 × 2 = **0**.4
    ...

- **In general, a fraction has a finite binary representation if the denominator has 2 as the only prime factor**

Activity 16 #3

Fixed-Point Representation

- A **fixed-point binary representation** uses a fixed number of bits *f* after the radix point to represent approximations to real numbers (actually, rational numbers)
  - Store *integers* in registers/memory, but treat the value as though the low *f* bits are fractional bits
  - Example: Suppose AL contains 00111011_2
    - The **underlying integer** value is 59 = 00111011_2
    - If $f = 4$, pretend there's a radix point to the right of bit 4, so AL represents $0011.1011_2 = 3\,^{11}/_{16}$
    - If $f = 2$, pretend there's a radix point to the right of bit 2, so AL represents $001110.11_2 = 14\,^{3}/_{4}$
  - Note: `float`/`double` variables in C/Java are different: use *floating-point* representation (future lecture)
- Fixed-point arithmetic can be done using *integer* arithmetic and bitwise operations
  - Useful for performance – integer operations are faster than floating-point operations
  - Also useful on processors that do not have a floating point unit
    - E.g., low-cost embedded microprocessors and microcontrollers

Q-Notation for Fixed Point Numbers

- Notation: **Q*f*** denotes a representation with *f* fractional bits
  - Q31 indicates a representation with 31 fractional bits (Q = "quantity of fractional bits")
  - Q15 indicates a representation with 15 fractional bits
- Alternative Notation:* **Q*m.f*** denotes a representation with
  - one sign bit
  - *m* integer ("magnitude") bits
  - *f* fractional bits
  - $\therefore$ Total number of bits is $1 + m + f$
- Examples: Suppose AL contains 00111011_2
  - Q4 or Q3.4 – Pretend there's a radix point to the right of bit 4, so AL represents $0011.1011_2 = 3\,^{11}/_{16}$
  - Q2 or Q5.2 – Pretend there's a radix point to the right of bit 2, so AL represents $001110.11_2 = 14\,^{3}/_{4}$

Activity 16 #4

*Not completely standard. Some people include the sign bit in *m*, so you might see Q16.16 instead of Q15.16. It's usually easy to figure out from context.

Scaling Factor

- The value of a fixed point data type is the value of the underlying integer, multiplied by a constant **scaling factor** (sometimes denoted by *S*)
  - Example: $1.110_2 = 1110_2 \times \mathbf{2^{-3}}$

- **A Q*m*.*f* fixed-point number has *f* fractional bits ⇒ scaling factor *S* = 2^{-f}**

- Examples: Suppose AL contains 00111011_2
  - Q4/Q3.4: Scaling factor is 2^{-4}, so AL represents $00111011_2 \times 2^{-4} = 0011.1011_2 = 3^{11}/_{16}$
  - Q2/Q5.2: Scaling factor is 2^{-2}, AL represents $00111011_2 \times 2^{-4} = 001110.11_2 = 14^3/_4$

- Scaling factor applies to negative numbers too
  - The 8 bits 11111111 form the 8-bit two's complement representation of **–1**
  - The 8 bits 11111111 form the 8-bit Q5.2 representation of $-1/4 = \mathbf{-1 \times 2^{-2}}$

Example: Q1.2

- Example:
  - Q1.2 ⇒ 4 total bits with 2 fractional bits
  - So scaling factor is $2^{-2} = {}^1/_4$
  - 4-bit two's complement integers and Q1.2 values shown:
  - Each Q1.2 integer is the corresponding integer value × 2^{-2}

Integer	Bits	Q1.2		
7	0111	$1^3/_4$	=	$^7/_4$
6	0110	$1^1/_2$	=	$^6/_4$
5	0101	$1^1/_4$	=	$^5/_4$
4	0100	1	=	$^4/_4$
3	0011	$^3/_4$	=	$^3/_4$
2	0010	$^1/_2$	=	$^2/_4$
1	0001	$^1/_4$	=	$^1/_4$
0	0000	0	=	$^0/_4$
–1	1111	$-^1/_4$	=	$-^1/_4$
–2	1110	$-^1/_2$	=	$-^2/_4$
–3	1101	$-^3/_4$	=	$-^3/_4$
–4	1100	–1	=	$-^4/_4$
–5	1011	$-1^1/_4$	=	$-^5/_4$
–6	1010	$-1^1/_2$	=	$-^6/_4$
–7	1001	$-1^3/_4$	=	$-^7/_4$
–8	1000	–2	=	$-^8/_4$

× scaling factor = $^1/_4$

Resolution & Range

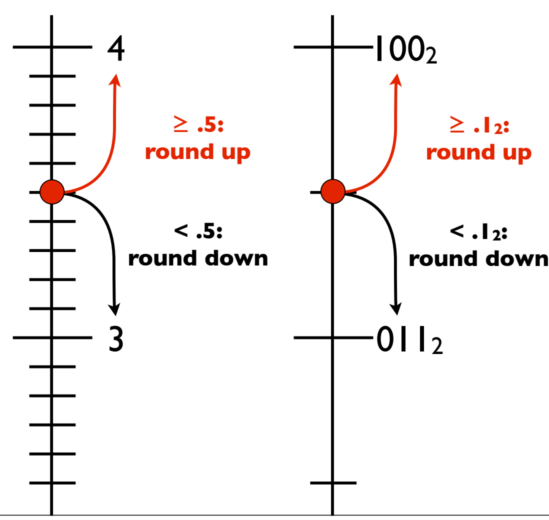
- One way to find the **range** of representable values (minimum/maximum values):
  - Find minimum and maximum for underlying integer type
  - Multiply by scaling factor
- The **resolution** ε is the smallest positive magnitude that can be represented
  - Example: Smallest Q2-representable positive number is $\varepsilon = .01_2 = 0.25$, so Q2 has a resolution of $^1/_4$
- In general, Q$m.f$ has
  - Range: $[-2^m,\ 2^m - 2^{-f}]$
  - Resolution: 2^{-f}

Integer	Bits	Q1.2		
7	0111	$1^3/_4$	=	$^7/_4$
6	0110	$1^1/_2$	=	$^6/_4$
5	0101	$1^1/_4$	=	$^5/_4$
4	0100	1	=	$^4/_4$
3	0011	$^3/_4$	=	$^3/_4$
2	0010	$^1/_2$	=	$^2/_4$
1	0001	$^1/_4$	=	$^1/_4$
0	0000	0	=	$^0/_4$
–1	1111	$-^1/_4$	=	$-^1/_4$
–2	1110	$-^1/_2$	=	$-^2/_4$
–3	1101	$-^3/_4$	=	$-^3/_4$
–4	1100	–1	=	$-^4/_4$
–5	1011	$-1^1/_4$	=	$-^5/_4$
–6	1010	$-1^1/_2$	=	$-^6/_4$
–7	1001	$-1^3/_4$	=	$-^7/_4$
–8	1000	–2	=	$-^8/_4$

Activity 16 #5

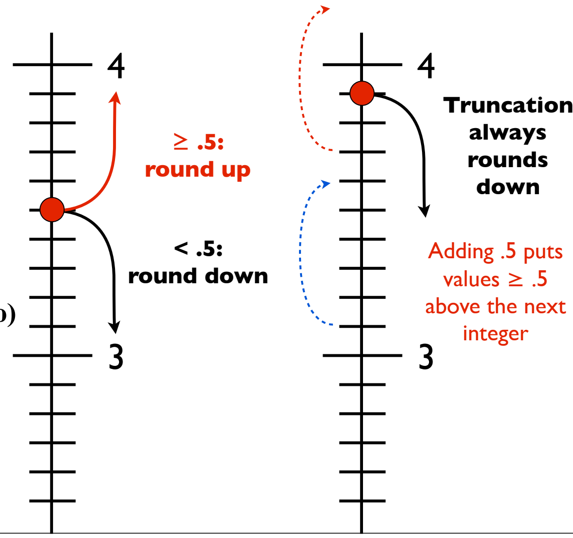
Rounding Fixed-Point Numbers (1)

- Recall rounding decimal numbers:
  - To nearest integer: 3.5 $\Rightarrow$ 4 3.4 $\Rightarrow$ 3
  - To nearest $^1/_{10}$: 3.75 $\Rightarrow$ 3.8 3.64 $\Rightarrow$ 3.6
  - Identify place you want to round to
  - Add 1 in that position if next digit is ≥ 5
  - Drop subsequent digits
- Rounding binary numbers is similar
  - Identify place you want to round to
  - **Add 1 in that position if the next bit is 1**
  - Drop subsequent bits
  - To nearest integer: 010.1 $\Rightarrow$ 011 010.0 $\Rightarrow$ 010
  - To nearest 1/2: 011.01 $\Rightarrow$ 011.1 011.11 $\Rightarrow$ 100.0



Rounding Fixed-Point Numbers (2)

- Equivalently, for decimal numbers:
  - Add .5, .05, .005, etc. ($^1/_2$ of unit to round to)
  - Drop subsequent digits
  - To nearest integer:
    3.5 + .5 = 4.0 ⇒ 4   3.4 + .5 = 3.9 ⇒ 3
  - To nearest 1/10:
    3.75 + .05 = 3.80 ⇒ 3.8   3.64 + .05 = 3.69 ⇒ 3.6
- Equivalently, for binary numbers:
  - **Add .1, .01, .001, etc. ($^1/_2$ of unit to round to)**
  - Drop subsequent bits
  - To nearest integer:
    011.1 + .1 = 100.0 ⇒ 100   011.0 + .1 = 011.1 ⇒ 011
  - To nearest 1/2:
    011.01 + .01 = 011.10 ⇒ 011.1



Conversion Between Scaling Factors

- To convert a fixed-point number with scaling factor R to one with scaling factor S:
  - Multiply the underlying integer value by $R \div S$
  - I.e., multiply by the ratio R / S
  - May require rounding (when converting to a representation with fewer fractional bits)

– Part 2: Arithmetic –

Addition & Subtraction

- Exercise:

000001.01_2	$1^1/_4$	00000101_2	5
$\underline{+\ 000000.01_2}$	$\underline{+\ ^1/_4}$	$\underline{+\ 00000001_2}$	$\underline{+\ 1}$

Activity 16 #7

- **To add/subtract fixed-point numbers, use integer addition/subtraction**
  - Use `add`, `sub` instructions as usual
  - Flags represent same conditions (overflow, carry, sign, zero) as for integer arithmetic
- Why does it work?
  - Suppose n_1 and n_2 are the underlying integers – they represent the values $n_1 \cdot 2^{-f}$ and $n_2 \cdot 2^{-f}$
  - $n_1 \cdot 2^{-f} + n_2 \cdot 2^{-f} = (n_1 + n_2) \cdot 2^{-f}$
  - So the sum of the integer values is the representation of the fixed-point sum

Multiplication

- Exercise:

$$\begin{array}{r} 000001.00_2 \\ \underline{\times\ 000001.00_2} \end{array} \qquad \begin{array}{r} 00000100_2 \\ \underline{\times\ 00000100_2} \end{array} \qquad \begin{array}{r} 1 \\ \underline{\times\ 1} \end{array}$$

Activity 16 #8

- **To multiply two Q*f* fixed-point numbers:**
  1. Multiply the underlying integers (`imul`), using twice as many bits to store the product
  2. If product is nonnegative, add $(1 \ll (f-1))$ to the product to ensure correct rounding
  3. Arithmetic right-shift by f bits
- **Summary:** $\boldsymbol{\big((a \times b) + (1 \ll (f-1))\big) \gg^s f}$
- Why does it work? (Ignoring rounding)
  - Suppose n_1 and n_2 are the underlying integers – they represent the values $n_1 \cdot 2^{-f}$ and $n_2 \cdot 2^{-f}$
  - $n_1 \cdot 2^{-f} \cdot n_2 \cdot 2^{-f} = n_1 \cdot n_2 \cdot 2^{-f} \cdot 2^{-f}$
  - Right-shift by f bits to multiply by 2^f, giving $= n_1 \cdot n_2 \cdot 2^{-f}$

Activity 16 #9

Division

- **To divide two Q*f* fixed-point numbers ($a \div b$):**
  1. Use twice as many bits to store the dividend (a), and left-shift it by f bits
  2. One way to ensure the quotient is correctly rounded (rounding away from 0):
     - If the quotient will be positive, add $b \div 2$ to the dividend
     - If the quotient will be negative, subtract $b \div 2$ from the dividend
  3. Perform integer division (`idiv`), returning the quotient
- **Summary:** $\boldsymbol{\big((a \ll f) + \mathrm{sgn}(a \div b) \cdot (b \gg^s 1)\big) \div b}$

$$\mathrm{sgn}(n) = \begin{cases} -1 & \text{if } n < 0 \\ 0 & \text{if } n = 0 \\ 1 & \text{if } n > 0 \end{cases}$$

- Why does it work? (Ignoring rounding)
  - Suppose n_1 and n_2 are the underlying integers – they represent the values $n_1 \cdot 2^{-f}$ and $n_2 \cdot 2^{-f}$
  - Left-shifting n_1 by f bits multiplies it by 2^f, so this represents $n_1 \cdot 2^{-2f}$
  - $\dfrac{n_1 \cdot 2^{-2f}}{n_2 \cdot 2^{-f}} = \dfrac{n_1 \cdot 2^f}{n_2 \cdot 2^{2f}} = \dfrac{n_1 \cdot 2^f}{n_2 \cdot 2^f} = n_1 \cdot n_2 \cdot 2^{-f}$

Activity 16 #10