EE 109 Appendix G

Emulating FP

USING INTEGERS TO EMULATE DECIMAL/FRACTION ARITHMETIC

FP Emulation

- Low-end microprocessors (like the Arduino) may not have hardware for floating point (`float` and `double`) operations
- If you do use `float` and `double` in your code, the compiler can include a large software library to emulate floating point operations with integer operations
  - Makes your program MUCH larger
  - NOT DESIRABLE if you can easily emulate FP operations with integer operations on your own

Option 1 for Performing FP

- Problem: Suppose you need to add, subtract and compare numbers representing FP numbers
  - Ex. amounts of money (dollars and cents.)
- Option 1: Use floating point variables
  - Pro: Easy to implement and easy to work with
  - Con: Some processors like Arduino don't have HW support of FP and so they would need to include a lot of code to emulate FP operations
  - Con: Numbers like $12.50 can be represented exactly but most numbers are approximate due to rounding of fractions that can be can't be represented in a finite number of bits.
    - Example 0.1 decimal can't be represented exactly in binary

```
float x, y, z;
if (x > y)
  z = z + x;
```

Option 1: Just use 'float' or 'double' variables in your code

Option 2 for Performing FP

- Option 2: Split the amounts into two integer variables, one for the dollars and one for the cents. $12.53 -> 12 and 53
  - Pro: Everything done by fixed-point operations, no FP.
  - Cons: All operations require at least two fixed point operations (though this is probably still faster than the code the compiler would include in Option 1)

```
/* add two numbers */
z_cents = x_cents + y_cents;
z_dollars = x_dollars + y_dollars;
if (z_cents > 100) {
  z_dollars++;
  z_cents -= 100;
}
```

Option 2: Use 'ints' to emulate FP

Option 3 for Performing FP

- Option 3: Use a single fixed-point variable by scaling (multiplying) the amounts by 100 (e.g. $12.53 -> 1253)
  - Pro: All adds, subtracts, and compares are done as a single fixed-point operation
  - Con: Must convert to this form on input and back to dollars and cents on output.

```
int x_amount = x_dollars*100+x_cents;
int y_amount = y_dollars*100+y_cents;

int z_amount;
if (x_amount > y_amount)
  z_amount += z_amount + x_amount;

int z_dollars = z_amount / 100;
int z_cents = z_amount % 100;
```

Option 3: Use single 'int' values that contain the combined values of integer and decimal

Emulating Floating Point

- Let's examine option 3 a bit more
- Suppose we have separate variables storing the integer and fraction part of a number separately
  - If we just added integer portions we'd get 18
  - If we account for the fractions into the inputs of our operation we would get 19.25 (which could then be truncated to 19)
  - We would prefer this more precise answer that includes the fractional parts in our inputs

Desired Decimal Operation (separate integer / fraction):

	Integer	Fraction
	12.	75
+	6.	5

Example and Procedure

- Imagine we took our numbers and multiplied them each by 100, performed the operation, then divided by 100
  - 100*(12.75+6.5) = 1275 + 650 = 1925 => 1925/100 = 19.25 => 19
- Procedure
  - Assemble int + frac pieces into 1 variable
    - Shift integer to the left, then add/OR in fraction bits
  - Peform desired arithmetic operation
  - Disassemble int + frac by shifting back

Desired Decimal Operation (separate integer / fraction):

	Integer	Fraction
	12.	75
+	6.	5

Shift Integers & Add in fractions:

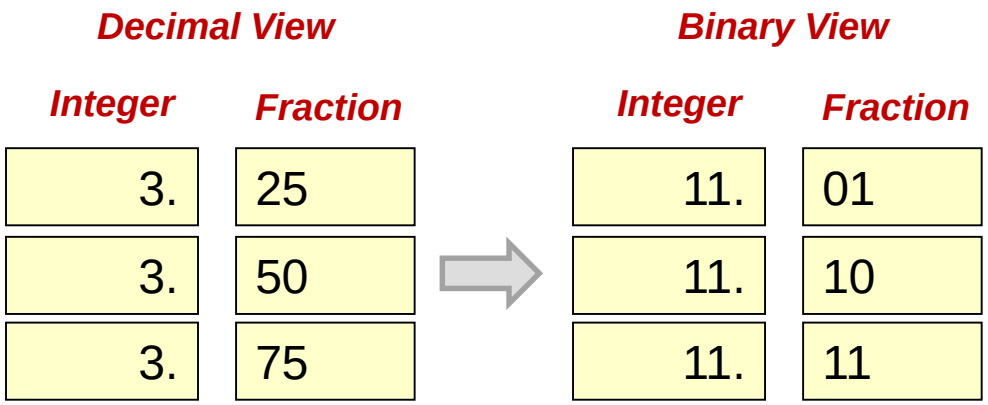
	1275.	0
+	650.	0
	1925.	0

Disassemble integer and fraction results (if only integer is desired, simply shift right to drop fraction)

Integer	Fraction	
19.	25	***Integer + Fraction***
19.		***Or just integer***

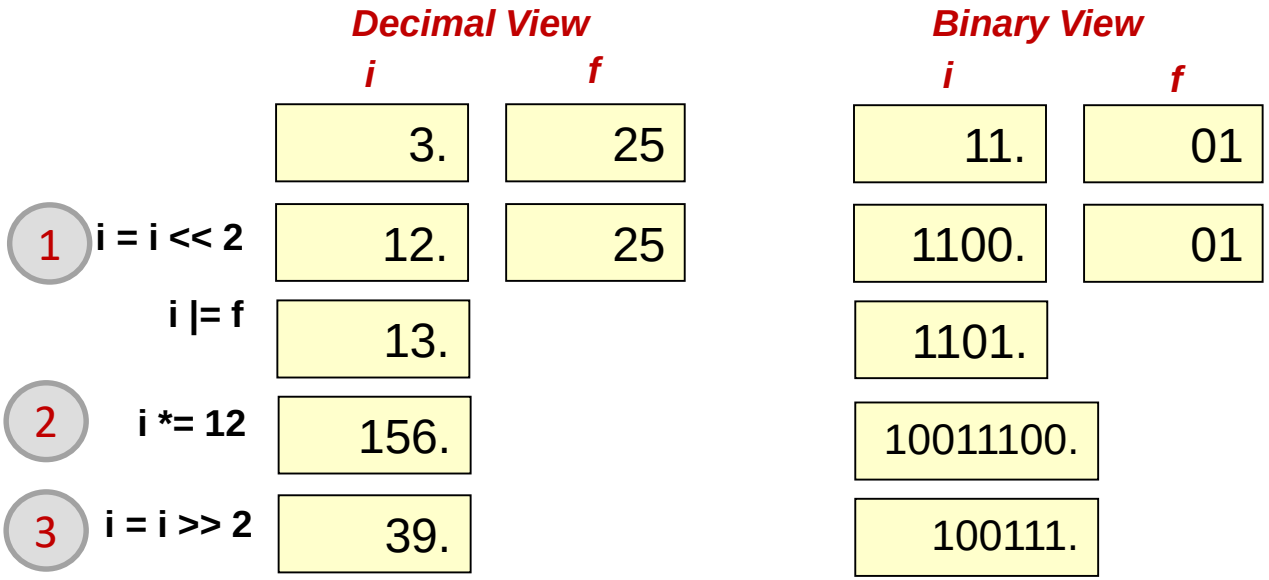
Example 2

- Suppose we want to convert from "dozens" to number of individual items (i.e. 1.25 dozen = 15 items)
  - Simple formula: 12*dozens = individual items
  - Suppose we only support fractions: ¼, ½, ¾ represented as follows:



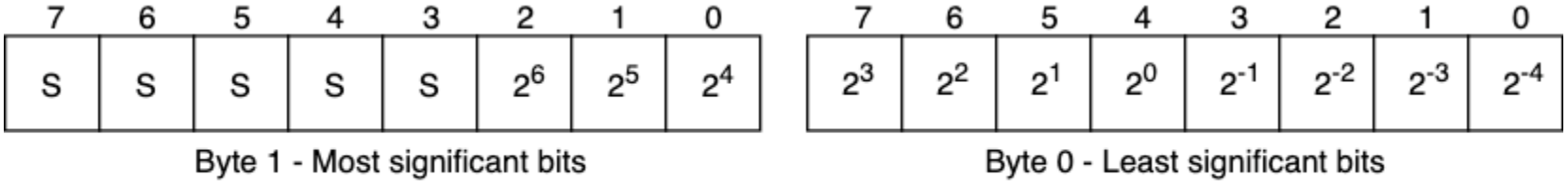
Example 2

- Procedure
  - Assemble int + frac pieces into 1 variable
    - Shift integer to the left, then add/OR in fraction bits
  - Peform desired arithmetic operation
  - Disassemble int + frac by shifting back



1-wire Temperature Sensor

- Returns (sign extended) 11-bit temperature reading accurate to 1/16 a degree Centigrade
- If you interpret these 16-bits as an integer, by what value is it implicitly scaled?
- Eventually, how can you unscale it?
- Before you unscale it, how can you use the techniques on the previous slides to get an answer that will be accurate to one-tenth of a degree Fahrenheit
- When should you add the constant 32?



Temperature values are returned in a two element array