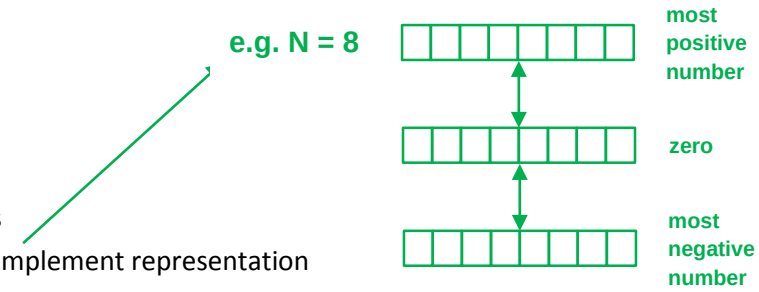
Chapter 3 – Fixed-Point vs Floating-Point Processing

Fixed-point and floating-point refer to the two basic kinds of number systems used in computers. There are two concepts to consider:

- numerical representation, i.e., how the bits in a datum are interpreted
- arithmetic operations on the data

The c2000 family includes processors that are fixed-point and processors that are floating-point.

1. Fixed-Point Processing



- numbers are integers
- most often use 2's complement representation
- arithmetic is done on integers

- integers are 16 bits on c2000
- long integers are 32 bits on c2000

- typically can achieve faster clock rate than floating-point
- typically processor has lower power consumption than floating-point
- typically processor is cheaper than floating-point processor

- programmer must be aware of limitations and may need to take steps to accommodate them:
  - introduce scaling
  - turn on "saturation mode"
  - implement "block floating-point"

2. Floating-Point Processing

- numbers are real
- floating-point representation
- arithmetic is done on real numbers

- regular floating-point numbers are 32 bits on c2000[1]

- floating-point has large dynamic range

- typically processor has higher power consumption than fixed-point
- typically processor is more expensive than fixed-point processor

- programmer has less concern about range limitations

[1] c2000 also supports “double” = 64 bits

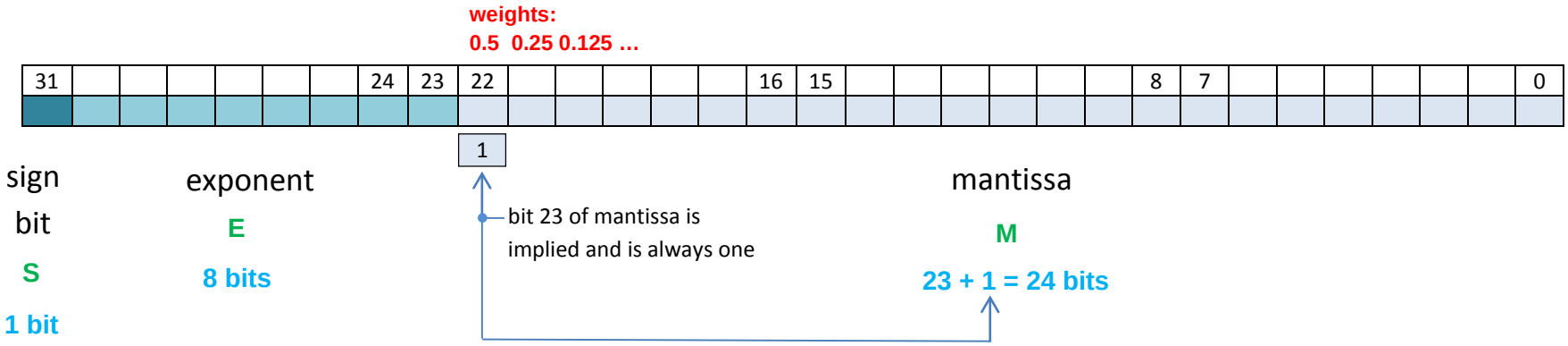
Fixed-Point Applications

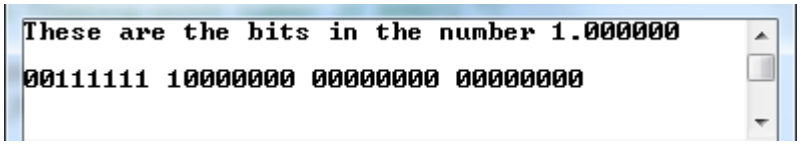
- good for “bit-exact” applications
- good for “control” parts of code:
    - if, then, else decisions
    - uses less memory than floating-point

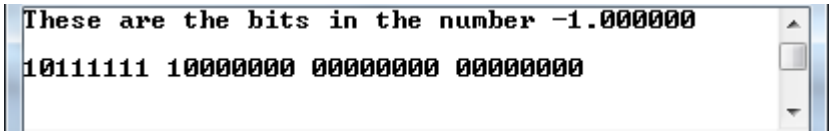
Floating-Point Applications

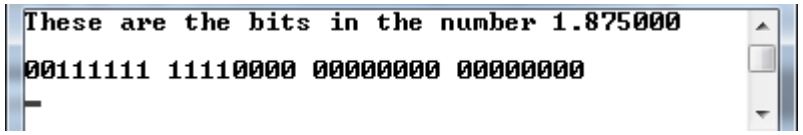
- good for applications in which the data have a large dynamic range

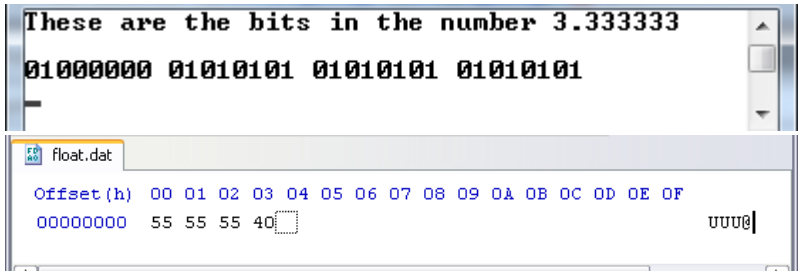
Floating-Point Number in Memory
Single-Precision 32-Bit `float`
IEEE 754 Standard Format

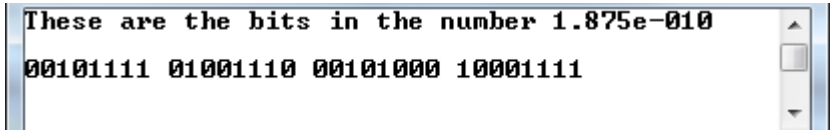



These are the bits in the number 1.000000
00111111 10000000 00000000 00000000


These are the bits in the number -1.000000
10111111 10000000 00000000 00000000


These are the bits in the number 1.875000
00111111 11110000 00000000 00000000


These are the bits in the number 3.333333
01000000 01010101 01010101 01010101
float.dat
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 55 55 55 40 UUU@


These are the bits in the number 1.875e-010
00101111 01001110 00101000 10001111

Special Cases in Floating-Point Format

Zero

0000 0000 0000 0000 0000 0000 0000 0000

or

1000 0000 0000 0000 0000 0000 0000 0000

E

(Note: Neither of these are a "real" zero as per the standard S1.M x 2^{E-127}.)

Infinity

0111 1111 1000 0000 0000 0000 0000 0000

or

1111 1111 1000 0000 0000 0000 0000 0000

E

"Not a Number" (NaN) e.g. 0/0 or x/0, i.e., result of divide-by-zero

S111 1111 1XXX XXXX XXXX XXXX XXXX XXXX

sign E anything but all 0's

Given the following 32-bit binary number written in standard floating-point format, what decimal number does it represent?

1100 0001 1010 0000 0000 0000 0000 0000

Given the following decimal number, write it as a binary number in standard floating-point format.

0.40625

Fixed-Point Saturation (assume 8-bit 2's complement numbers)

Consider adding these two numbers:

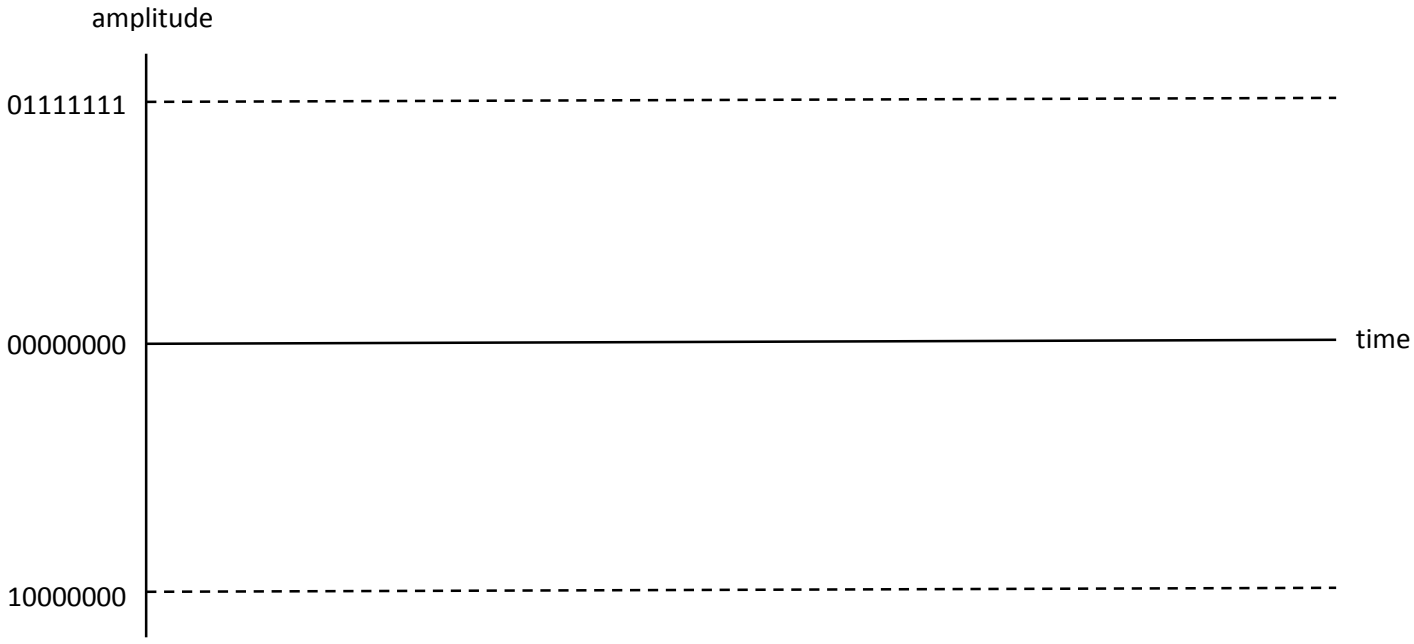
0	1	1	1	1	1	0	0
+							
0	0	0	0	0	0	0	1

Now consider adding these two numbers:

0	1	1	1	1	1	1	1
+							
0	0	0	0	0	0	0	1

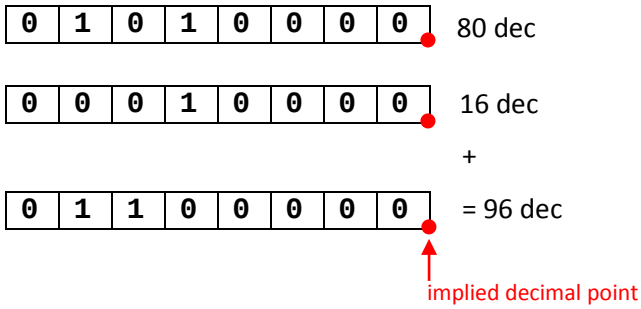
Note that fixed-point addition can potentially overflow, but fixed-point multiplication will not.
Note that floating-point addition can also potentially overflow, but it is much more unlikely.

Example of saturation – high-voltage sine-wave output signal



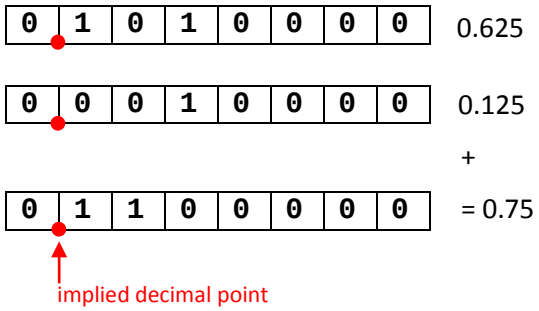
Some Examples of Fixed- and Floating-Point Additions and Multiplications

Fixed-Point Add Operation – Integer Interpretation

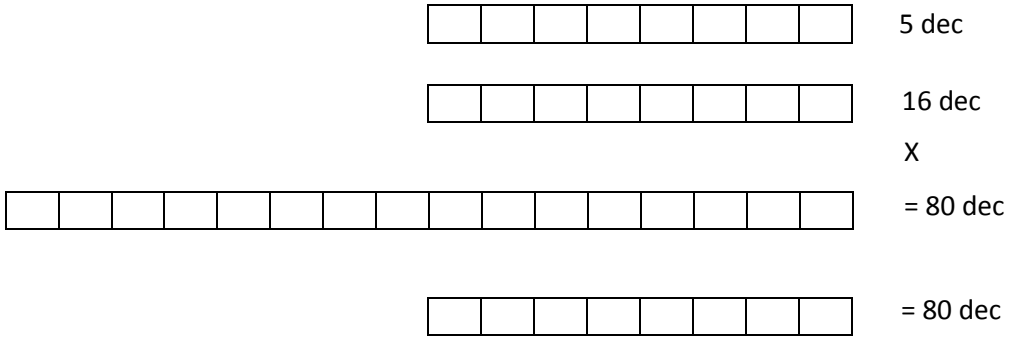


look up Q format in book or Internet

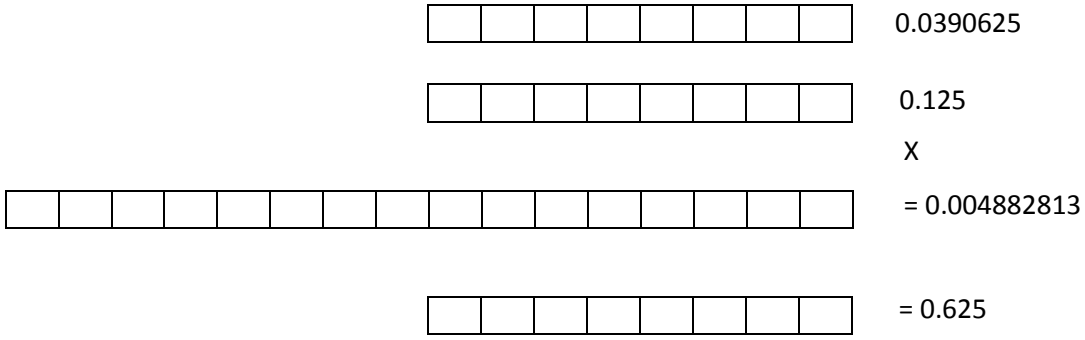
Fixed-Point Add Operation – Q7 Interpretation



Fixed-Point Multiply Operation – Integer Interpretation



Fixed-Point Multiply Operation – Q7 Interpretation



Floating-Point Add Operation

12.0 | 1.5 | x 2^3 | + | 0.4375 | 1.75 | x 2^{-2}

= 12.4375

Floating-Point Multiply Operation

12.0 | 1.5 | x 2^3 | X | 0.4375 | 1.75 | x 2^{-2}

= 5.25