

p 520 Converting to Floating-point numbers (10)

- Given a decimal number, we convert to floating point using:

1. Convert the decimal into binary.
2. Normalize the binary number.
3. Calculate the biased exponent.

4. Combine 1, 2, and 3 to store the number into floating point format.

Lecture #3

1. Algorithm for converting from decimal into binary form

Break every decimal into two: integer-part, fractional-part.

For the integer-part, we follow the procedure that we outlined before:

Quotient	Remainder	digit
$\text{int}(N/2) = q_0$	$N - 2q_0 = r_0$	$r_0$
$\text{int}(q_0/2) = q_1$	$q_0 - 2q_1 = r_1$	$r_1$
$\text{int}(q_1/2) = q_2$	$q_1 - 2q_2 = r_2$	$r_2$
$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$

until  $q_{n+1}$  becomes zero.

The resulting binary form is:

$$r_n r_{n-1} \dots r_2 r_1 r_0$$

for the integral part of the number (skip the following)

Perhaps an easier way to remember the algorithm:

Quotient	Remainder	Digit
$\text{int}(N/2) = q_0$	<u>If</u> (N is odd) <u>then</u> $r_0 = 1$ <u>else</u> $r_0 = 0$	$r_0$

$\text{int}(q_0/2) = q_1$	<u>If</u> ( $q_0$ is odd) <u>then</u> $r_1 = 1$ <u>else</u> $r_1 = 0$	$r_1$
---------------------------	---	-------

$\text{int}(q_1/2) = q_2$	<u>If</u> ( $q_1$ is odd) <u>then</u> $r_2 = 1$ <u>else</u> $r_2 = 0$	$r_2$
---------------------------	---	-------

⋮	⋮	⋮
---	---	---

until  $q_{n+1} = 0$

This can be generalized for converting the fractional part.

Let the fractional part be  $f_0$ .

Then, we can compute the binary representation using:

number	integral-part	fractional-part	digit
$2 * f_0$	$\text{int}(2 * f_0) = i_0$	$2 * f_0 - i_0 = f_1$	$i_0$
$2 * f_1$	$\text{int}(2 * f_1) = i_1$	$2 * f_1 - i_1 = f_2$	$i_1$
$2 * f_2$	$\text{int}(2 * f_2) = i_2$	$2 * f_2 - i_2 = f_3$	$i_2$
$2 * f_3$	$\text{int}(2 * f_3) = i_3$	$2 * f_3 - i_3 = f_4$	$i_3$

until either (i)  $f_n$  is zero, or (ii) we have computed as many digits as we need.

For converting into an arbitrary radix  $m$ , simply replace 2 by  $m$  in the table above.   
 → See examples on the back

(13)

Example :  $0.625_{10} = (??)_2$

$2 \times 0.625 = 1.250 \rightarrow l_0 = 1$

$2 \times 0.250 = 0.5 \rightarrow l_1 = 0$

$2 \times 0.5 = 1.0 \rightarrow l_2 = 1$

$\rightarrow 0.625_{10} = 0.101_2$

$= (0.5 + 0.125)_{10}$

↑ Example:

$28.21_{10} = (??)_{16}$

$28 \quad 12 \rightarrow C$

$1 \quad 1 \rightarrow 1$

$28_{10} = 1C_{16}$

$16 \times 0.21 = 3.36 \quad 3$

$16 \times 0.36 = 5.76 \quad 5$

$16 \times 0.76 = 12.16 \quad C$

$16 \times 16 = 0.0256 \quad 0$

$.21_{10} \approx 35C0_{16}$

$\approx 20996 \dots_{10}$

$(28.21)_{10} \approx (1C.35C0)_{16}$

# Why does it work?

- Consider a number written in an arbitrary radix  $m$ :

$$N = r_n \dots r_3 r_2 r_1 r_0 . i_0 i_1 i_2 \dots \leftarrow \text{this expansion can be infinite.}$$

with integral-part:

$$N_i = r_n r_{n-1} \dots r_3 r_2 r_1 r_0$$

and fractional-part:

$$N_f = .i_0 i_1 i_2 \dots$$

(Then, clearly,

$$q_0 = \text{int} \left[ \frac{r_n m^n + r_{n-1} m^{n-1} + \dots + r_1 m + r_0}{m} \right]$$
$$= r_n m^{n-1} + r_{n-2} m^{n-2} + \dots + r_1$$

and  $N_i - q_0 \cdot m = r_0$ , the least significant digit.

It is also clear that repeating the rest of the digits. (for the integral part)

skip this

15

For the fractional part:

$$\begin{aligned} \text{int}(m * N_f) &= \text{int} [ (i_0 \cdot m^{-1} + i_1 \cdot m^{-2} + \dots), m ] \\ &= i_0, \text{ the most significant} \\ &\quad \text{digit of the fractional} \\ &\quad \text{part.} \end{aligned}$$

As before, it is clear how to repeat the steps.

Example:  $1/10$  cannot be represented <sup>exactly</sup> in base 2. (16)

We have:

$$2 * 0.1 = 0.2 \Rightarrow i_0 = 0$$

$$2 * 0.2 = 0.4 \Rightarrow i_1 = 0$$

$$2 * 0.4 = 0.8 \Rightarrow i_2 = 0$$

$$2 * 0.8 = 1.6 \Rightarrow i_3 = 1$$

$$2 * 0.6 = 1.2 \Rightarrow i_4 = 1$$

$2 * 0.1 = 0.2$  which puts us back to  $i_0 = 0$ .

The whole sequence repeats.

$$\begin{aligned} 0.1 &= 0.00011 \overline{00011} \\ &= 0.\overline{00011} \end{aligned}$$

$\Rightarrow$  It would take an infinite number of bits to represent 0.1

If 0.1 represents \$0.1, a dime, it will take us an infinite number of bits to represent a dime on the computer

Unless we represent numbers in base ten!  
Floating point representation.

# Arithmetic Operations in different bases

These are essentially the same as in decimal, ...

For displaying memory in the debugger, we need to understand hex.

Consider addition in hex:

$$\begin{array}{r}
 11 \leftarrow \text{"carry"} \\
 3FE \\
 + \quad E2 \\
 \hline
 4E0
 \end{array}$$

How?

$$\begin{aligned}
 (E + 2)_{16} &= (14 + 2)_{10} = (16)_{10} \\
 &= (1 \cdot 16^1 + 0 \cdot 16^0)_{16} \\
 &= (10)_{16}
 \end{aligned}$$

So, write down 0 for the sum, and then carry the 1 for the next addition.

$$\begin{aligned}
 (1 + F + E)_{16} &= (1 + 15 + 14)_{10} \\
 &= (16 + 14)_{10} \\
 &= (1 \cdot 16^1 + 14 \cdot 16^0)_{10}
 \end{aligned}$$

→ See back!!!

$$= 1E \leftarrow \text{write E, carry 1}$$

(18)

$$\begin{array}{r} A34_{16} \\ \times 26_{16} \\ \hline 3D38 \\ 1468 \quad + \\ \hline 183B8_{16} \end{array}$$

①  $6_{16} \times 4_{16} = 6_{10} \times 4_{10} = 24_{10} = 18_{16}$

write 8, carry 1.

②  $6_{16} \times 3_{16} + 1_{16} = 6_{10} \times 3_{10} + 1_{10} = 18_{10} + 1_{10} = 19_{10} = 13_{16}$

write 3, carry 1

③  $6_{16} \times A_{16} + 1_{16} = 6_{10} \times 10_{10} + 1_{10} = 61_{10} = 3D_{16}$

write 3D

$$\begin{array}{r}
 A\ 3\ 4 \\
 -\quad 2\ 6 \\
 \hline
 A\ 0\ (E)
 \end{array}$$

How?

① 4 is smaller than 6, borrow  $1 \cdot 16^1$ :  
 $(4 + 1 \cdot 16 - 6)_{10} = (14)_{10}$   
 $= E_{16}$

Write down (E), borrow (1).

②  $3 - 2 - (1)^* = 0$

Write down 0, no borrowing.

③  $A - 0 = A$ .

For binary addition/subtraction, the situation is similar, but we borrow 2 (more on this coming!).

Binary addition:

$$\begin{array}{r}
 1\ 1 \\
 1011 \\
 1101 \\
 0101 \\
 1111 + \\
 \hline
 101100
 \end{array}$$

Binary multiplication:

$$\begin{array}{r}
 1011 \\
 \times 1101 \\
 \hline
 1011 \\
 0000 \\
 1011 \\
 1011 \\
 \hline
 1000111
 \end{array}$$

$$\begin{array}{r}
 1 \\
 10 \\
 1 \\
 10 \\
 10 \\
 10 \\
 100
 \end{array}$$

Binary subtraction:

$$\begin{array}{r}
 1101 \\
 -1011 \\
 \hline
 0010
 \end{array}$$

# Binary Coded Decimals (BCDs) (Devry, P33)

## I] Packed BCD numbers.

Represent each decimal digit using 4 bits (same as hexadecimal). Range of numbers 0 to 9  $0000_2$   $1001_2$

Example:

$$(45)_{10} = \overset{8421}{(0100)_2} \cdot 10^1 + \overset{8421}{(0101)_2} \cdot 1$$

$$= (0100 \ 0101) \text{ packed BCD}$$

## II] Unpacked BCD numbers

Not needed

Represent each decimal digit using 8 bits.

Example:

$$(45)_{10} = (\underbrace{0000}_{\text{always zero}} \ 0100 \ \underbrace{0000}_{\text{always zero}} \ 0101)_{\text{unpacked BCD}}$$

# American Standard Code for Information Interchange

ASCII (see pages 271-274 of the blue book) 21  
USED TO REPRESENT STRINGS

Codes Represent  
Control characters:

00 → 1f h

eg: 0d h is CR (carriage return)  
0ah is LF (line-feed)

20h is a blank-space

⋮	0
30h	1
31h	2
32h	3
33h	⋮
⋮	9
39h	

Lowest four-bits  
represent the second  
digit in the ascii code:  
eg 2 in 32h,  
and they also  
represent the actual  
number.

→ 30h + N is the ascii code for N.  
(0 ≤ N ≤ 9)

41h	A
42h	B
⋮	⋮
5Ah	Z
61h	a
62h	b
⋮	⋮
7Ah	Z

Adding 20h to  
the ASCII code  
of an upper  
case letter would  
convert it to a lower  
case letter.

→ See back!!!