

COMP20005 Workshop Week 9

1
2

LAB

Discussion 1: Representation of integer
Discussion 2: Representation of float

- Array exercises: Consolidate your understanding of arrays, by completing Exercises 7.7, 7.8, and 7.15. *Arrays, and being able to manipulate data stored in them, are the absolute backbone of this subject.*

Assignment 2:

- released this week, due Fri week 11
- be ready for discussion in workshop week 10

Converting Numbers from Decimal → Binary

Exercise: Write the following numbers in the binary and hexadecimal formats:

Decimal format	Binary Format	Hexadecimal format
9		
55		
5.875		

Notes:

- better to convert the integer and the fractional parts separately
- one hexadecimal digit is equivalent to 4 binary digits
- remember the powers of two:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
128	64	32	16	8	4	2	1	0.5	0.25	0.125

Twos-compliment representation of integers using w bits

- Note that we use a fixed amount of bits w
- w bits gives 2^w combinations \rightarrow possible range : $-2^{w-1} \dots + (2^{w-1}-1)$
 - -128 to $+127$ if $w=8$
 - -2^{31} to $+2^{31}-1$ if $w=32$

Representation of a signed integers :

- Positive number x : as the binary representation of x in w bits. Note: the first bit is always 0
- Negative numbers x : as the *twos-complement* of (the representation) of $|x|$ in w bits. The first bit will always be 1 and has the value -2^{w-1}

A trick: Finding twos-complement representation in w bits

Suppose that we need to find the twos-complement representation of a negative x in $w=16$ bits. Do it in 3 steps:

- 1) Write binary representation of $|x|$ in w bits
- 2) Find the **rightmost one-bit**
- 3) Inverse (ie. flip 1 to 0, 0 to 1) all bits **on the left** of that **rightmost one-bit**

find the 2-comp repr of -36	Bit sequence			
1) bin repr of 36 in 16 bits	0000	0000	0010	0100
2) find the rightmost 1-bit	0000	0000	0010	0100
3) inverse the left of that 1-bit	1111	1111	1101	1100

→ -36 is represented as the bit pattern 1111 1111 1101 1100,
or FFDC in the hexadecimal format

Understand why the above twos-complement format is used!

Exercise 1: 2-complement representation in w=16 bits

Q: What are 17, -17, 34, and -34 as 16-bit twos-complement binary numbers, when written as (a) binary digits, and (b) hexadecimal digits?

Decimal	twos-complement as binary digits	twos-complement as hexadecimal digits
17		
-17		
34		
-34		

Representation of floats

We learnt 2 different formats:

- one as described in [numericA.pdf](#) and in the text book
- another is an [IEEE standard](#), which is:
 - employed in most of modern computers,
 - demonstrated in the lecture, and
 - you can find/experiment with using the program [floatbits.c](#) ([numericA.pdf](#) p.22).

COM
P100
02.W
orksh
op A

Representation of float x: principle

Convert $|x|$ to binary form, and transform so that

- Option a: $|x| = 0.b_0b_1b_2\dots \times 2^e$ where $b_0 = 1$
- Option b: $|x| = 1.b_1b_2\dots \times 2^e$ where the integer part $b_0 = 1$

so $|x|$ can be represented as 2 integers (e, m) where:

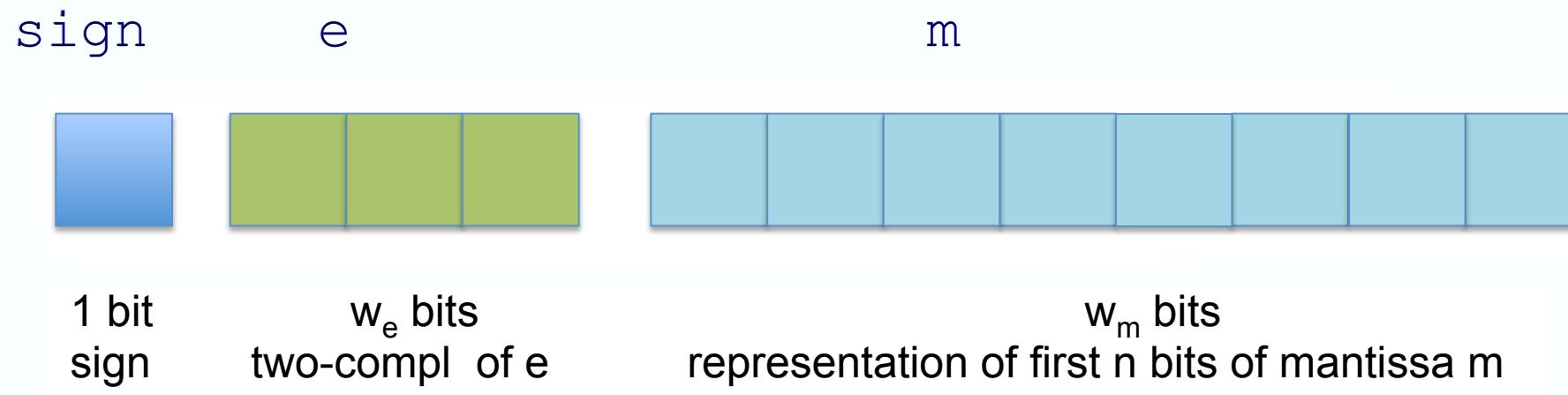
- for Option a: $m = b_0b_1b_2\dots$
- for Option a: $m = b_1b_2\dots$ (which hidden b_0 since b_0 is always 1)

Then, x is represented as triple $(\text{sign}, \text{exponent } e, \text{mantissa } m)$, and:

- using 1 bit for sign (0: positive, 1:negative)
- using w_e bit for exponent (note e can be negative)
- using w_m bits for mantissa (with possible cutting off right bits or adding 0-bits to the right of mantissa to fit into w_m bits)

Note: There are different standards on representing float. Each is characterized by the value of w_e and w_m , the representation of e , and by the choice of mantissa (with or without b_0). There is no need to remember the details of any standard, but the knowledge of w_e and w_m could be helpful.

Example 1: Representation of floats as described in numericA.pdf



Convert $|x|$ to binary form, and transform so that:

$$|x| = 0.b_0b_1b_2\dots \times 2^e \text{ where } b_0 = 1$$

$$m = b_0b_1b_2\dots$$

Note: just an example, not important, no need to remember the above details!

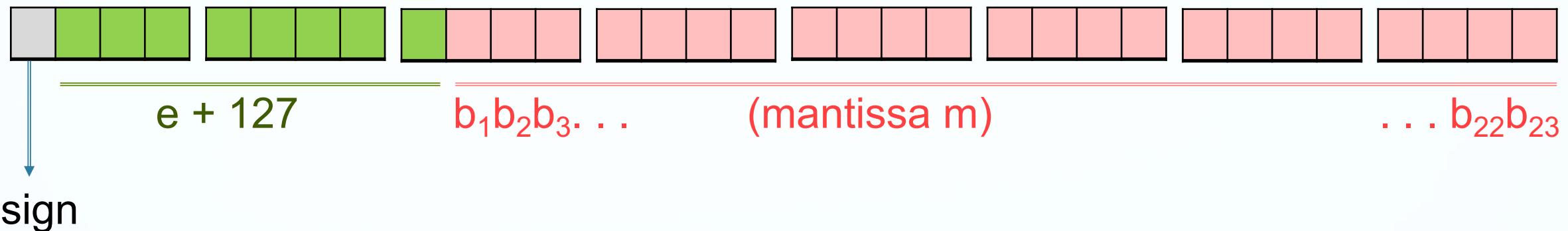
Exercise 2

Class/Group Exercise: using the described 12-bit representation (page 21 of the numericA.pdf slides), what are the floating point representations of -3.125, 17.5, and 0.09375, when written as (a) binary digits, and (b) hexadecimal digits?

Do it, then check your answer with your classmates or tutor:

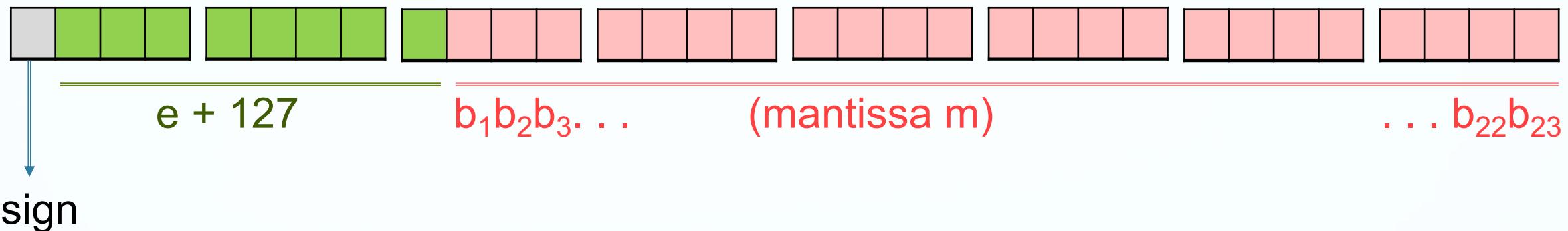
value	Binary	Hexadecimal
-3.125		
17.5		
0.09375		

Example 2: Representation of 32-bit float: (IEEE 754, as in floatbits.c)



- using $w_e = 8$ bit for e :
 - e is represented as the (unsigned) bit pattern for the value $127+e$
 - so:
 - $e= 5$ is represented as ???
 - $e= -2$ is represented as ???
- using $w_m = 23$ bits for m

Representation of 32-bit float: (IEEE 754, as in floatbits.c)



Important Note:

- remembering that $w_e=8$, and that the bit pattern for $e=0$ is 0111 1111 and/or for $e=1$ (1000 0000) could be helpful!

Un-important Notes, just for fun:

- we say that e is represented in **excess-127** format
- Valid e is -126 → +126, corresponding to the bit pattern 0000 0001 → 1111 1110 .
- zero is 32 **zero-bit**, and infinity is 32 **one-bit**.

Representation of 32-bit float: (IEEE 754, as in floatbits.c)

$$w_s=1, w_e=8, w_m=23 \quad |x| = 1.b_1b_2\dots \times 2^e$$

Example: $x = 6.5$

In binary: $x = 110.1 = 1.101 \times 2^2$

→ sign bit: 0
→ $e = 2 \Rightarrow 127 + 2 = 128 + 1$ is represented as 1000 0000 + 1

 → e is represented as 1000 0001

→ mantissa: 101 00...0

→ Final representation:

0100 0000 1101 0000 0000 0000 0000

or 4 0 D 0 0 0 0 (16)

Exercise 2 (modified)

Class/Group Exercise: using representation in modern PC (an IEEE standard), what are the floating point representations of -3.125, 17.5, and 0.09375, when written as (a) binary digits, and (b) hexadecimal digits?

Note: this question was modified from the original question, which as for the 12-bit floating point representation described in the textbook.

Do it, then check your answer with your classmates or tutor:

value	Binary	Hexadecimal
-3.125		
17.5		
0.09375		

- transform binary number to $1.\text{???} \times 2^e$
- w=32, $\textcolor{blue}{w_e}=8$, the bit pattern for $e=0$ is $0111\ 1111$, for $e=1$ is $1000\ 0000$

2D arrays

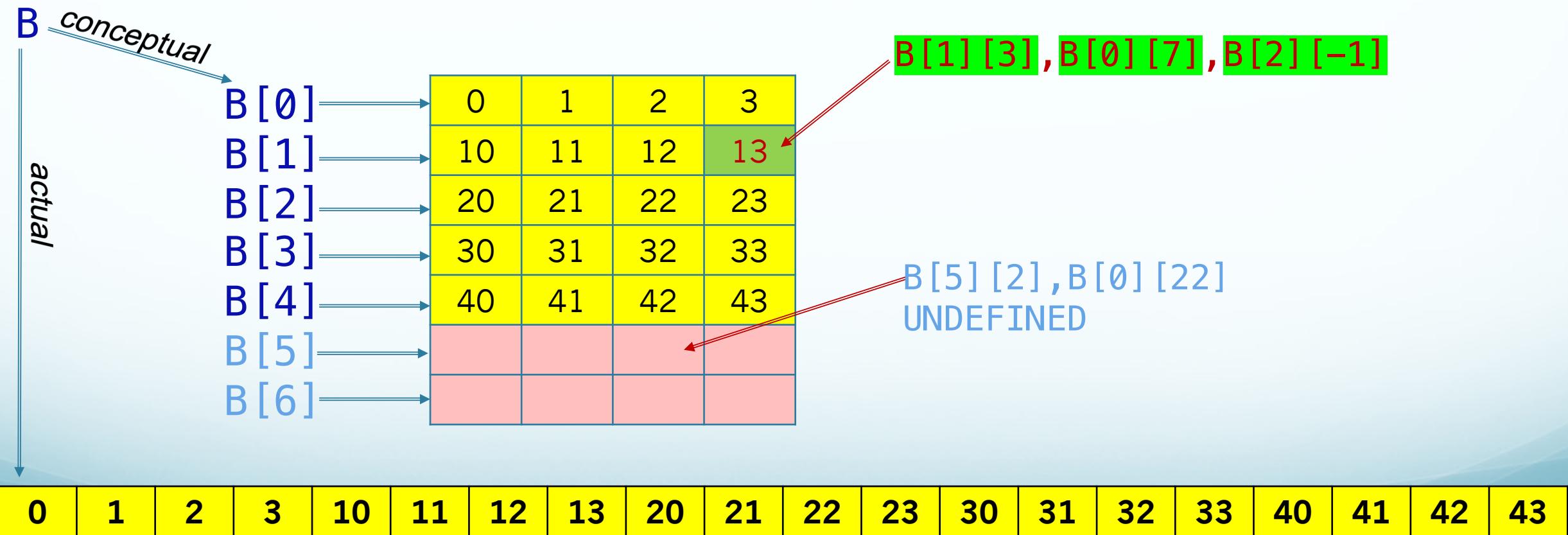
Each data type has a corresponding pointer type, for example

```
int A[5],  B[3][4];
//      B is an array of      "array of 4 int"
//      B is a pointer-to      pointer-to-int
int *pa, **pb;
//      pb is a "pointer-to-pointer-to-int"
pa = A;           ✓
pa = A+3;         ✓
pb = B;           ✓
pb = &B[0];        ✓
pb = &B[0][0];     ✗
A = A+3;         ✗
```

if A is an 1D array, i is an int, A[i] is valid, but might be undefined!
if B is an 2D array, B[i][j] is valid, but might be undefined!

```
int B[5][4]={{0,1,2,3},{10,11,12,13},{20,21,22,23},{30,31,32,33},{40,41,42,43}};
```

- B is an array of arrays, each B[i] is an array (and a pointer)
- B[i] points to B[i][0], ie. B[i] has the value of &B[i][0]



example

```
int A[5]={4,3,5,2,1};
```

```
int B[3][5]
```

```
={{1,12,3,14,5},{15,4,6,2,13},{10,7,8,11,9}};
```

What's the value of

1. B[A[4]][A[1]]
2. B[A[1]][A[2]]
3. B[A[4]][A[3]]
4. B[A[3]][A[2]]
5. B[A[3]][-7]

COM
P100
02.W
orksh
op A

LAB

- Manipulating arrays, including:
 - Array exercises: completing Exercises 7.7, 7.8, 7.03, and 7.15.
 - 7.07: find most frequent value
 - 7.08: find k-smallest
 - 7.03: sort & remove duplicates
 - 7.15: anagrams
 - Other exercises in Chapter 7
 - Remember: *Arrays, and being able to manipulate data stored in them, are the absolute backbone of this subject.*
- [If you want a challenge] Write a program that outputs the bit-patterns associated with 64-bit double values.

Additional Material

- Searching in Arrays
- Procedures for converting between from decimal to binary representation

A Task: Searching in Arrays

General task: given an array of n elements A[n], find a specific element.

Example:

1. Find a special value, such as:
 - a) being equal to x, ie. find i such that $A[i]==x$
 - b) find the value of k-smallest, or k-largest element
2. Find some “optimized” value, such as:
 - a) find the maximal/minimal element
 - b) Find the most frequent element

Searching for a specific element

Task: search for a particular value in an array

Examples

- Return index i such that $A[i] == x$ (or return -1 if NOTFOUND): *there is a chance for NOTFOUND*
- Exercise 7.08 (k-smallest)

```
#define NOTFOUND -1

// return the first index i
int search(int A[], int n) {
    ???
    for (i=0; i<n; i++) {
        // process A[i], test if A[i] is the solution
        //
        // return i or A[i] if YES
    }
    return NOTFOUND;
}
```

Searching for an “optimized” element

Optimization Task: Example

- search for the maximal (or minimal) value in an array.
- exercise 7.07 (most-frequent)

Note: *solution always found, ie. there is normally no NOTFOUND case*

```
// return a kind of min/max value in A
// pay attention when there are multiple soln

int find(int A[], int i) {
    // soln= first estimation
    for (i=0; i<n; i++) {
        // process A[i]:
        // test if A[i] is a better solution
        // if YES, update soln
    }
    return soln;
}
```

Procedure for Decimal→Binary: Integer Part

Changing integer x to binary (when the solution not obvious):

*Just divide x and the subsequent quotients by 2 until getting zero. The sequence of remainders, in **reverse order** of appearance, is the binary form of x.* 6567

Example: 23

operation	quotoiion	remainder
$23 : 2$	11	1
$11 : 2$	5	1
$5 : 2$	2	1
$2 : 2$	1	0
$1 : 2$	0	1

So: $23 = 10111_{(2)}$ $11 = \underline{1}_{(2)}$ $46 = \underline{\underline{1}}_{(2)}$

Procedure for Decimal→Binary: Fraction Part

Fraction: Multiply it, and subsequent fractions, by 2 until getting zero. Result= sequence of integer parts of results, in appearance order. Examples: 0.1+01+... +0.1

0.375			0.1		
operation	int	fraction	operation	int	fraction
.375 x 2	0	.75	.1 x 2	0	.2
.75 x 2	1	.5	.2 x 2	0	.4
.5 x 2	1	.0	.4 x 2	0	.8
			.8 x 2	1	.6
			.6 x 2	1	.2

$$\text{So: } 0.375 = 0.\textcolor{red}{011}_{(2)}$$

$$0.1 = 0.\textcolor{red}{00011(0011)}_{(2)}$$