COMP 250

Lecture 2

Binary number representations

Sept. 9, 2016

Base 10 (decimal) "digits" {0,1,2,..., 9}

e.g. $5819 = 5 * 10^3 + 8 * 10^2 + 1 * 10^1 + 9 * 10^0$

Base 2 (binary) "bits" {0, 1}

e.g.

$$(11010)_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$$

decimal	binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
:	:

decimal	binary
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010
11	00001011
:	:

Fixed number of bits (typically 8, 16, 32, 64)

8 bits is called a "byte".

To converting from binary to decimal, you need to know the powers of 2.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
:	:

$$(11010)_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$$

$$= 16 + 8 + 0 + 2 + 0$$

$$= 26$$

How to convert from decimal to binary?

$$(241)_{10} = (\ ?\ )_2$$

I will present an algorithm for doing so, but let's first review some basic ideas.

$$m = (m/10) * 10 + m \,\%\, 10$$

I changed the following slightly after the lecture.

(Integer) division by 10 = dropping rightmost digit

238 / 10 = 23

Multiplication by 10 = shifting left by one digit

23 * 10 = 230

Remainder of integer division by 10 = rightmost digit

238 % 10 = 8

Claim: the same idea works is binary.

$$m = m/2 * 2 + m \,\%\, 2$$

e.g.

$$m = (10011)_2$$

$$m/2 = (1001)_2$$

$$(m/2) * 2 = (10010)_2$$

$$m \,\%\, 2 = (00001)_2$$

$$m = (b_{n-1} b_{n-2} \cdots b_2 b_1 b_0)_2 = \sum_{i=0}^{n-1} b_i 2^i$$

$$m \% 2 = b_0$$

$$m/2 = (b_{n-1} b_{n-2} \cdots b_2 b_1)_2 = \sum_{i=1}^{n-1} b_i 2^{i-1}$$

Algorithm: convert from decimal to binary

```
i ← 0
while   m > 0 {
    b[i] ← m % 2
    m  ←  m / 2
    i  ←  i + 1
}
```

Example

241

120	1	b_0
60	0	b_1
30	0	b_2
15	0	
7	1	etc.
3	1	
1	1	
0	1	

Thus $241 = (11110001)_2$

Addition in binary

$$\begin{array}{r} 11010 \\ +\quad 1111 \\ \hline ? \end{array} \qquad\qquad \begin{array}{r} 26 \\ +\quad 15 \\ \hline 41 \end{array}$$

Addition in binary

carry	11110		
	11010		26
+	1111	+	15
	101001		41

Grade school arithmetic in binary

- addition
- subtraction
- multiplication
- division

There is nothing special about base 10.

These algorithms work for binary (base 2) too.

How many bits N do we need to represent an integer n?

$$n = \sum_{i=0}^{N-1} b_i \, 2^i$$

That is, what is the relationship between n and N ?

To answer this question, we use:

$$2^N - 1 = \sum_{i=0}^{N-1} 2^i$$

$$= (2^{N-1} + 2^{N-2} + 2^{N-3} + \ldots$$

$$\ldots + 16 + 8 + 4 + 2 + 1)$$

which is a special case of the following, where $x = 2$.

$$\frac{x^N - 1}{x - 1} = 1 + x + x^2 + x^3 + \ldots \; x^{N-1}$$

How many bits N do we need to represent an integer n?

$$n = \sum_{i=0}^{N-1} b_i \, 2^i \quad \leq \sum_{i=0}^{N-1} 2^i$$

$$= \quad 2^N - 1 \qquad \text{(previous slide)}$$

$$< \quad 2^N$$

Taking the log (base 2) of both sides:

$$\boxed{\log_2 n \; < \; N}$$

How many bits N do we need to represent an integer n?

$$n = \sum_{i=0}^{N-1} b_i \, 2^i$$

We can assume that $b_{N-1} = 1.$

e.g. We ignore leftmost 0's of $(00000010011)_2$

$$n = \sum_{i=0}^{N-1} b_i \, 2^i \geq 2^{N-1}$$

Taking the log (base 2) of both sides:

$$\boxed{\log_2 n \geq N - 1}$$

Thus,

$$\log_2 n < N \leq (\log_2 n) + 1$$

$$log_2 n \; < \; N \; \leq \; (log_2 n) + 1$$

Q: How many bits N do we ***need*** to represent n?

A: The largest integer less than or equal to $(log_2 n) + 1.$

We write:

$$N = floor(\,(log_2 n) + 1)$$

where "floor" means "round down".

n (decimal)	n (binary)	$N = floor\,(1 + \; log_2\, n)$
0	0	
1	1	1
2	10	2
3	11	2
4	100	3
5	101	3
6	110	3
7	111	3
8	1000	4
9	1001	4
10	1010	4
11	1011	4
:	:	:

The number of bits N that we need to represent an integer n is

$$N = floor((log_2 n) + 1)$$

which is O($log_2 n$).

BTW, we will use the above formula several times throughout the course.

I cannot remember it, so I don’t expect you to either.

What about other ‘bases’ ?

$$n = \sum_{i=0}^{N-1} a_i \ base^i$$

$$n = (a_{N-1} \ a_{N-2} \ \ldots \ a_1 \, a_0)_{base}$$

where $0 \leq a_i < base - 1$

The arithmetic algorithms +,-,*,/ all work for arbitrary bases.
So does the base conversion algorithm (which we saw only for binary).

$$m = (m/b) * b + m \% b$$

Note that this is equation we saw last lecture:

$$a = q * b + r$$

Other number representations

(covered in detail in COMP 273 –
see my lecture notes if you are curious)

Q: How many bits are used to represent integers in a computer?

Q: How are negative integers represented ?

Q: How are non-integers (fractional numbers) represented ?