

Lecture Notes on Number Systems

1. Question: Which is the bigger number?

5 3 2 (ans 5)

Alternative question: Which is “five”?

five V cinq 101 (ans none)

The point: These are all names of the abstract notion of ‘fiveness’. We can invent LOTS of other systems to name numbers.

NUMBER != NUMERAL

2. Positional (Polynomial) Number Systems

Example: decimal: $346 = 3 \cdot 100 + 4 \cdot 10 + 6 \cdot 1$

general: “*radix* R” or “base R”:

$$\text{Integers } d_n d_{n-1} \dots d_0 = \sum_{i=0}^n d_i \times R^i$$

and

$$\text{Reals } d_n d_{n-1} \dots d_0 \bullet d_{-1} d_{-2} \dots d_{-m} = \sum_{i=-m}^n d_i \times R^i$$

↑
Radix Point

- The “d’s” are generally 0, 1, ... R
- Common notation: ddd_R

Do some examples: *binary*, ternary, *octal*, *hexadecimal*.

3. Conversion

Radix R - to - Decimal: Compute the expansion

$$n = d_n R^n + \dots + d_0 R^0$$

Decimal - to - Radix R

$$\frac{n}{R} = d_n R^{n-1} + \dots + d_1 R^0, \text{ with remainder } d_0$$

Algorithm:

- Divide successively by R.
- Note remainder at each step. (order of digits is lsb to msb)

Radix R_1 to Radix R_2

R_1 to decimal to R_2

4. Systems of Special Interest

<i>Binary</i>	base 2	Easy for computers
<i>Octal</i>	base 8	Easier for people
<i>Hexadecimal</i>	base 16	Easier for people

Show conversion between all of these

Emphasize hex numbering 0 ... 9ABCDEF

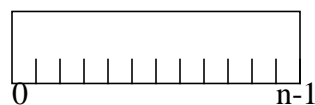
5. Integer Representation in Real Computers

Issues: **Fixed Size, Negative Number Representation**

A. Fixed Size

“The XXX is a 16-bit (vs 32-bit) computer ...”

If “word” size is n bits



there are 2^n possible bit patterns so only 2^n possible distinct numbers

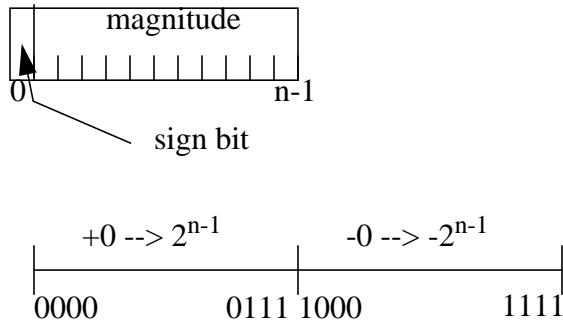
Implications:

- Cannot represent all possible numbers!!
- Must use some of these bit patterns (half?) to represent negative numbers.

B. Negative Number Representation

i. sign magnitude (the obvious one)

- reserve one bit for the sign
- rest of bits are interpreted directly as the number



Notes:

- Two (2) zeros - generally bad.
- Arithmetic costly (we'll see).

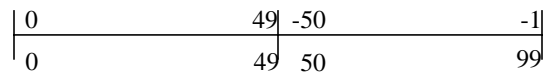
ii. radix and radix-1 complement (also called r's and (r-1)'s)

-k represented as $R^n - k$

example: base 10 -- 10's complement, $n=2$.

$+3 = +3$

$-3 = 10^2 - 3 = 97$



Negation is simple: "Complement" every digit and add 1.

$100 - k = 99 - k + 1$

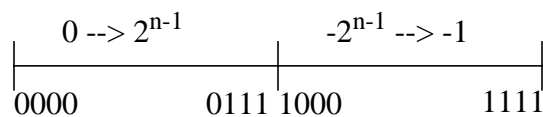
$$\begin{array}{r} 99 + 1 \\ -xy \\ \hline \overline{xy} + 1 \end{array}$$

Example ($R=10$): $-(38) = 99 - 38 + 1 = 61 + 1 = 62$

$-(62) = 99 - 62 + 1 = 38$

2's complement (special case):

$-k = 2^n - k$



Negation: Complement means flip the bit, so flip every bit and add one

Example (R=2, n=8):

```

    9510      0010111
    flip      1101000
    add 1     1101001  (-95)
  
```

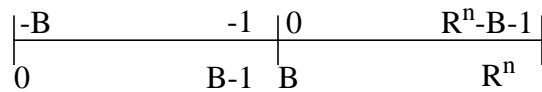
Note that “re-negating” gets us back to the original.

Notes:

- One zero (this is good)
- Arithmetic easy (we’ll see)

iii. excess B

Represent k (whether + or -) as k + B



Note: allows asymmetric range.

6. Arithmetic on Positional Systems

A. General Arithmetic

We all know how to do arithmetic in radix 10 using carries

854 +172 --- 1026	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;">0</td> <td style="padding: 5px;">1</td> <td style="padding: 5px;">2</td> <td style="padding: 5px;">...</td> <td style="padding: 5px;">9</td> </tr> <tr> <td style="padding: 5px;">0</td> <td style="padding: 5px;">0,0</td> <td style="padding: 5px;">1,0</td> <td style="padding: 5px;">2,0</td> <td style="padding: 5px;"></td> <td style="padding: 5px;">9,0</td> </tr> <tr> <td style="padding: 5px;">1</td> <td style="padding: 5px;">1,0</td> <td style="padding: 5px;">2,0</td> <td style="padding: 5px;">3,0</td> <td style="padding: 5px;"></td> <td style="padding: 5px;">0,1</td> </tr> <tr> <td style="padding: 5px;">2</td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;">9</td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;"><sum,carry></td> </tr> </table>		0	1	2	...	9	0	0,0	1,0	2,0		9,0	1	1,0	2,0	3,0		0,1	2						9					<sum,carry>
	0	1	2	...	9																										
0	0,0	1,0	2,0		9,0																										
1	1,0	2,0	3,0		0,1																										
2																															
9					<sum,carry>																										

Same in other radices, just use a different table.

Example: Our hero, binary.

010011 +001011 --- 011110	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;">0</td> <td style="padding: 5px;">1</td> </tr> <tr> <td style="padding: 5px;">0</td> <td style="padding: 5px;">0,0</td> <td style="padding: 5px;">1,0</td> </tr> <tr> <td style="padding: 5px;">1</td> <td style="padding: 5px;">1,0</td> <td style="padding: 5px;">0,1</td> </tr> </table>		0	1	0	0,0	1,0	1	1,0	0,1	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;">0</td> <td style="padding: 5px;">1</td> </tr> <tr> <td style="padding: 5px;">0</td> <td style="padding: 5px;">1,0</td> <td style="padding: 5px;">0,1</td> </tr> <tr> <td style="padding: 5px;">1</td> <td style="padding: 5px;">0,1</td> <td style="padding: 5px;">1,1</td> </tr> </table>		0	1	0	1,0	0,1	1	0,1	1,1
	0	1																		
0	0,0	1,0																		
1	1,0	0,1																		
	0	1																		
0	1,0	0,1																		
1	0,1	1,1																		

B. Sign magnitude

- Add the operands, then check the signs and the magnitudes of the operands to determine the sign of the result.
- It’s a pain and expensive to do in hardware, lots of comparisons and control required.

C. 2's complement

- Addition and subtraction are easy. Add the two numbers and throw away the carry.
- Don't need to worry about the signs of the operands.

Examples : Assume 4-bit computer, first bit is sign bit.

1000	(-8)	1101	(-3)	1111	(-1)
<u>+0001</u>	(+1)	<u>+0010</u>	(+2)	<u>+0001</u>	(+1)
1001	(-7)	1111	(-1)	<u>±0000</u>	(0)

D. Overflow

i. Demonstration

OY!! That's Negative??!

0110	(+ 6)
<u>+0111</u>	(+ 7)
1101	(+13)

We call this **OVERFLOW**.

The number "13" is not representable in a 4-bit, 2's complement computer.

Question: Does the problem go away if we have more bits?

Answer: Nope, not in general, though we could safely add 6+7 with more bits.

ii. Detection.

- Examine the signs of the operands and the result.
- If signs of operands are different, cannot have overflow.
- If sign of operands are the same and the sign of the result is different, **overflow!!**
- These checks are easy to do in hardware.

7. Representation of Other Scalars

A. Alphanumeric characters

8-bit byte

7-bit ASCII code¹

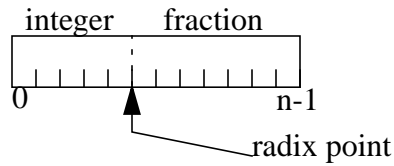
'a' = 61 ₁₆	'A' = 41 ₁₆	',' = 3B ₁₆
'z' = 7A ₁₆	'Z' = 5A ₁₆	' ' = 20 ₁₆

B. Real numbers

i. Fixed Point

- Given n bit number, radix point is "fixed" at some bit f
- So f "integer" bits and n-f "fractional" bits

1. ASCII -- American Standard Code for Information Interchange.



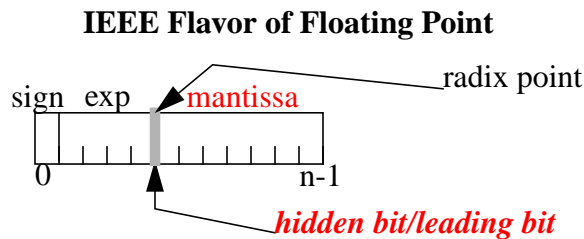
- Fixed Point not used because of limited range.

ii. Floating Point

Computer analog of “scientific notation”.

Instead of 547.764, $5.47764 * 10^2$

IEEE standard specifies number of bits for fraction (*mantissa*) and *exponent*, plus a bunch of other stuff



Examples:

Assume 12 bit computer, 7 (8 w/ hidden) mantissa, 4 exponent, excess 8

0 1010 1010000

exp: $10 - 8 = 2$

mantissa: $1.101 = 1 + 1/2 + 1/8 = 1.625$

$1.625 * 2^2 = 6.5$

1 0101 0110000

exp: $5 - 8 = -3$

mantissa: $1.011 = 1 + 1/4 + 1/8 = 1.375$

$1.375 * 2^{-3} = 0.171875$

Notes:

- 4 sizes: single, single extended, double, double extended
- exponent is excess B
- $R = 2$
- generally normalized (hidden bit is always 1)
- “*denormalized*” numbers possible for very small numbers (exponent is minimal and fractional bits are shifted right. Continued shifting causes “*gradual underflow*”).
- hidden bit not stored
- special bit patterns in exponent for NaN and infinity

- single representation of zero -- exponent and fraction are all 0

Table 1: IEEE Floating Point Standard.

	Single	Double
Mantissa bits(incl hidden)	24	53
sign bits	1	1
exponent bits	8	11
Bias/Excess	127	1023
Max Exponent	127	1023
Min Exponent	-126	-1022

8. IMPORTANT!!

Question: What's the value of "0110010110111001"?

Answer: You CANNOT tell. There is no notion of TYPE at the bit level in computers. These notions are realized at higher levels.