Representation of `int` data

World Inside a Computer is Binary

Decimal Number System

- Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- *Radix-10* positional number system. The radix is also called the *base* of the number system.

$$12304 = 1\times10^4+2\times10^3+3\times10^2+0\times10^1+4\times10^0$$

Quintinary Number System

- Digits[a]: 0, 1, 2, 3, 4
- *Radix-5* positional number system.

$$12304 = 1 \times 5^4 + 2 \times 5^3 + 3 \times 5^2 + 0 \times 5^1 + 4 \times 5^0$$

The value is 954 in decimal.

---

[a] Is quintinary a English work?

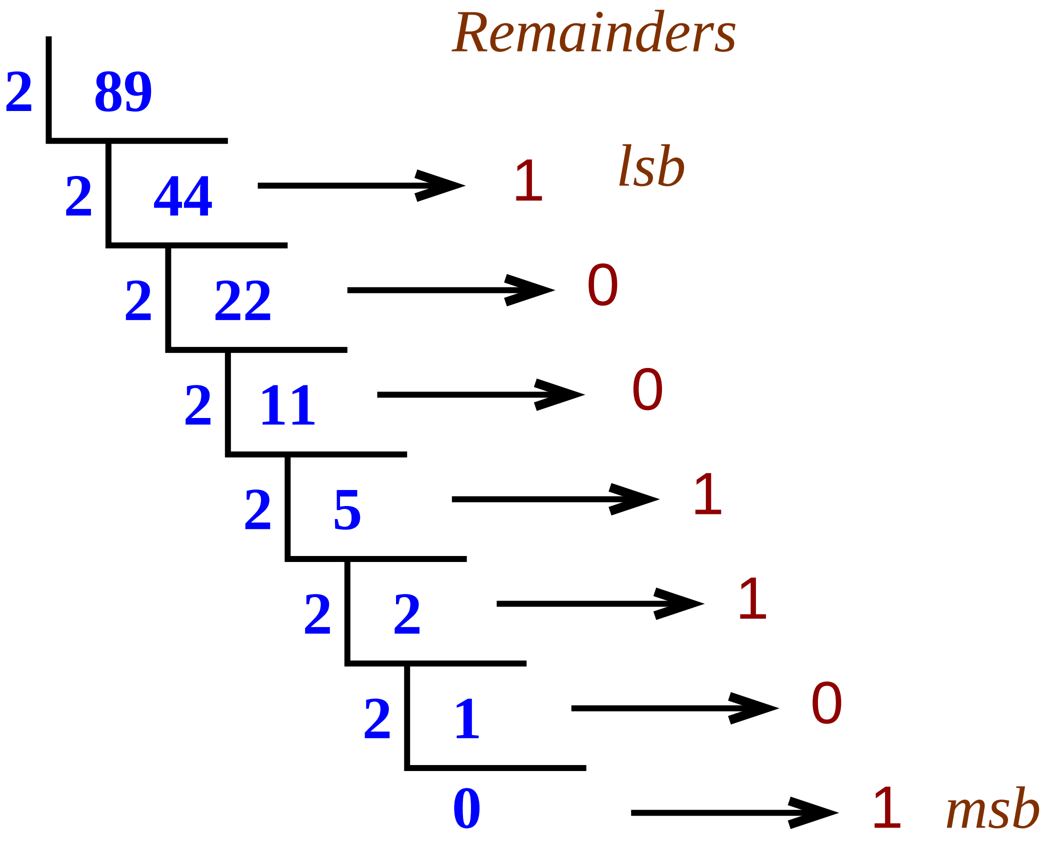
Unsigned Binary Number System

- Digits: 0, 1
- *Radix-2* positional number system.

$$10110 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

The value is 22 in decimal.

Decimal to Binary Conversion



Remainders

2 | 89

2 | 44 → 1 *lsb*

2 | 22 → 0

2 | 11 → 0

2 | 5 → 1

2 | 2 → 1

2 | 1 → 0

0 → 1 *msb*

Decimal 89 is equivalent to 1 0 1 1 0 0 1

Decimal to Binary Conversion

$$\begin{aligned} 89_D &= 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 2 \cdot (2 \cdot (2 \cdot (2 \cdot (2 \cdot (2 \cdot 1 + 0) + 1) + 1) + 0) + 0) + 1 \end{aligned}$$

Fixed and Finite Word

Every CPU can process (*add*, *subtract* etc.) a fixed length binary number by its machine instruction. Typical sizes of this data for a modern CPU are 16-bit, 32-bit, 64-bit etc. This is called the word size of a CPU. It is 32-bit for a Pentium processor[a].

---

[a] There are 64-bit versions and it can also process 8 and 16-bits data.

An Example with 4-bit Word

b_3	b_2	b_1	b_0

The Range of unsigned integer stored in 4-bits is:

$$\boxed{0 \text{ to } 2^4 - 1 = 15}$$

Bit String				*Decimal*
b_3	b_2	b_1	b_0	*Value*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

Bit String				*Decimal Value*
b_3	b_2	b_1	b_0	
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

`unsigned int` is 32-bits in C

31	30		0
		$\cdots$	

- Largest Number: $\sum_{i=0}^{31} 2^i = 4294967295$,

31	30		0
1	1	$\cdots$	1

- Smallest Number: 0

31	30		0
0	0	$\cdots$	0

Signed Decimal Number

In mathematics we use '$+$' and '$-$' symbols to indicate sign of a number.
But within a computer only two symbols $\{0, 1\}$ are available to represent any information. So one extra bit is required to indicate the sign of a number.

Three Popular Schemes

Signed Magnitude, 1's Complement and 2's Complement methods.
We shall only discuss about the *2's complement* representation of integers.

2's Complement Method

Consider a 4-bit word as an example.

b_3	b_2	b_1	b_0

b_3 is zero (0) for a positive number and it is one (1) for a negative number.
For an n-bit representation, the bit b_{n-1} represents the sign of the number.

2's Complement Numeral

$b_3 = 0$: remaining three bits give the value of the positive number.

Bit String				*Decimal*
b_3	b_2	b_1	b_0	*Value*
0	0	0	0	0
0	0	0	1	+1
0	0	1	0	+2
0	0	1	1	+3
0	1	0	0	+4
0	1	0	1	+5
0	1	1	0	+6
0	1	1	1	+7

Negative Numeral

If n is in 2's complement form, then $-n$ is obtained by taking the 2's complement of n!

2’s Complement of n

The 2’s complement of n is obtained by changing every bit of n to its complement (1’s complement) and finally adding 1 to the number.

Bit String				*Decimal*
b_3	b_2	b_1	b_0	*Value*
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

2's Complement Numeral

A negative number has a one (1) in the most significant position.

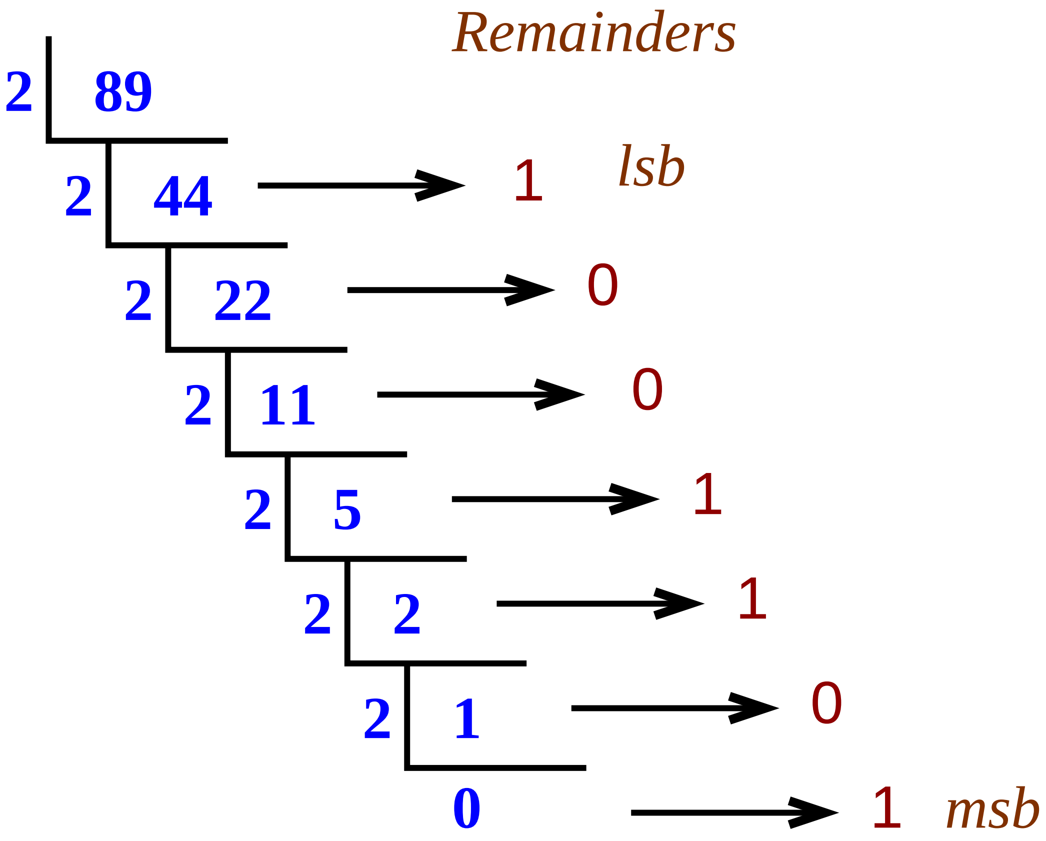
$$\overbrace{0\ 1\ 1\ 0\ 1\ 1\ 0\ 1}^{+109} \xrightarrow{\text{1's complement}} 10010010 + 1$$

$$\xrightarrow{\text{2's complement}} \overbrace{1\ 0\ 0\ 1\ 0\ 0\ 1\ 1}^{-109}$$

$$\overbrace{1\ 0\ 0\ 1\ 0\ 0\ 1\ 1}^{-109} \xrightarrow{\text{1's complement}} 01101100 + 1$$

$$\xrightarrow{\text{2's complement}} \overbrace{0\ 1\ 1\ 0\ 1\ 1\ 0\ 1}^{+109}$$

Direct 2's Complement



Remainders

2 | 89

2 | 44 → 1 *lsb*

2 | 22 → 0

2 | 11 → 0

2 | 5 → 1

2 | 2 → 1

2 | 1 → 0

0 → 1 *msb*

Decimal 89 is equivalent to 1 0 1 1 0 0 1

2's Complement Numeral

- Unique representations of zero: 0000.
- The range is $[-8 \cdots + 7]$ for 4-bits.
- For n-bits, the range is $[-(2^{n-1}) \cdots + (2^{n-1} - 1)]$.

`int, short int`

1. The range of data of type `int` (32-bits) is `-2147483648` to `2147483647`.
2. The range of `short int` (16-bits) is `-32768` to `32767`.
3. The range of `long long int` (64-bits) is `-9223372036854775808` to `9223372036854775807`.

2's Complement Addition

$$\begin{array}{rr|rr} & 0011 & & 3 \\ + & 0010 & + & 2 \\ \hline & 0101 & & 5 \end{array}$$

$$\begin{array}{rr|rr} & 1101 & & -3 \\ + & 1110 & + & -2 \\ \hline \cancel{1} & 1011 & & -5 \end{array}$$

$$\begin{array}{rr|rr} & 0011 & & 3 \\ + & 1011 & + & -5 \\ \hline & 1110 & & -2 \end{array}$$

$$\begin{array}{rr|rr} & 0100 & & 4 \\ + & 0101 & + & 5 \\ \hline & 1001 & & -7 \end{array}$$

Overflow

2's Complement Addition

$$\begin{array}{rr|rr} & 1101 & & -3 \\ + & 1010 & + & -6 \\ \hline 1 & 0111 & & 7 \end{array}$$

Overflow

`int` of C in Your Machine

The `int` in your machine (gcc-Linux on Pentium) is a 32-bit 2's complement number. Its range is -2147483648 to $+2147483647$. If one (1) is added to the largest positive number, the result is the smallest negative number.

0111 1111 1111 1111 1111 1111 1111 1111 +1	2147483647
1000 0000 0000 0000 0000 0000 0000 0000	−2147483648

Python

Python directly supports arithmetic on multi-precision data. In C there is multi-precision data library gmp.

Fibonacci Numbers

A Fibonacci number is an infinite sequence of integers where the 0^{th} element is 0, 1^{st} element is 1, and any i^{th} element $F_i,\quad i > 1$ is $F_{i-1} + F_{i-2}$. A few initial terms of the sequence are,

i	0	1	2	3	4	5	6	7	$\cdots$
F_i	0	1	1	2	3	5	8	13	$\cdots$

Assignment 13

Write a Python program that reads a non-negative integer n and computes the n^{th} Fibonacci number.

Assignment 14

Write a Python program that reads a positive integer n and draws a square whose each side has n $*$'s. As an example, if $n = 4$, then the output is

```
****
*  *
*  *
****
```

Assignment 15

Write a Python program that reads a positive integer n and prints its binary representation.
Input: 13
Output: 1101 Note: `str(x)` makes a string when `x` is a number.

Assignment 16

Write a Python program that reads a string of decimal digits and converts it to an integer corresponding to the reverse of the digit string.
Input: ”18763653”
Output: 35636781 Note: A string of decimal digits x can be converted to integer using `int(x)`.