计算机组成原理
Principles of Computer Organization

第 7 讲 计算机中数的编码与表示IV

浮点数规格化；IEEE754浮点数表示

主讲教师：石 侃
shikan@ict.ac.cn

2025年3月17日

3. 浮点数的规格化形式

6.2

$r = 2$	尾数最高位为 1	
$r = 4$	尾数最高 2 位不全为 0	基数不同，浮点数的
$r = 8$	尾数最高 3 位不全为 0	规格化形式不同

4. 浮点数的规格化

$r = 2$	左规	尾数左移 1 位，阶码减 1
	右规	尾数右移 1 位，阶码加 1
$r = 4$	左规	尾数左移 2 位，阶码减 1
	右规	尾数右移 2 位，阶码加 1
$r = 8$	左规	尾数左移 3 位，阶码减 1
	右规	尾数右移 3 位，阶码加 1

基数 r 越大，可表示的浮点数的范围越大

基数 r 越大，浮点数的精度降低

例如：16位浮点数，设 $m = 4$，$n = 10$，$r = 2$ 6.2

尾数规格化后的浮点数表示范围

最大正数 $2^{+1111} \times 0.\underbrace{1111111111}_{10\text{个}1} = 2^{15} \times (1-2^{-10})$

最小正数 $2^{-1111} \times 0.1\underbrace{000000000}_{9\text{个}0} = 2^{-15} \times 2^{-1} = 2^{-16}$

规格化

最大负数 $2^{-1111} \times (-0.1\underbrace{000000000}_{9\text{个}0}) = -2^{-15} \times 2^{-1} = -2^{-16}$

规格化

最小负数 $2^{+1111} \times (-0.\underbrace{1111111111}_{10\text{个}1}) = -2^{15} \times (1-2^{-10})$

浮点数(Floating Point)的表示范围

例：画出下述32位浮点数格式的规格化数的表示范围。

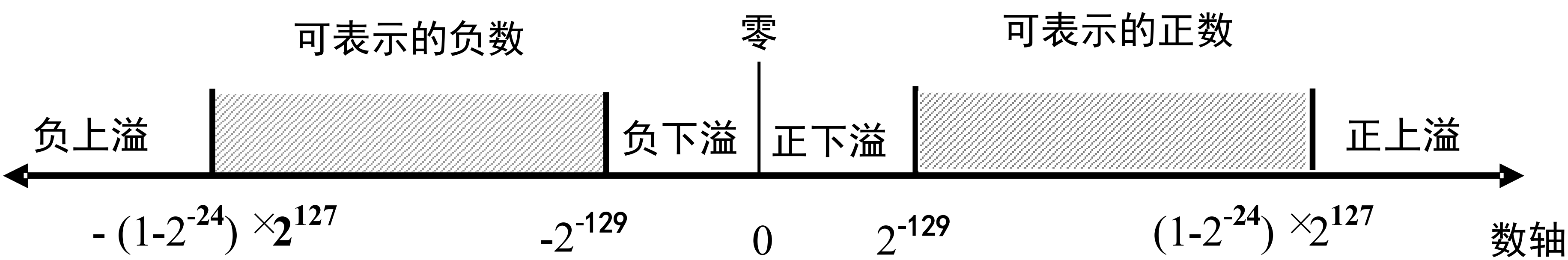
0	1　　　8	9　　　31
S_f	阶码j	尾数S

+/- 0.1xxxxx × 2 ^{j}

第0位数符S_f；第1～8位为8位"移码"表示阶码j（偏置常数为128）；第9～31位代表24位二进制原码小数表示的尾数S。规格化尾数的小数点后第一位总是1，故规定第一位默认的"1"不明显表示出来。这样可用23个数位表示24位尾数。

最大正数：$0.11\ldots1 \times 2^{11\ldots1} = (1-2^{-24}) \times 2^{127}$　最小正数：$0.10\ldots0 \times 2^{00\ldots0} = (2^{-1}) \times 2^{-128}$

因为原码是对称的，所以其表示范围关于原点"零"对称。



机器零：尾数为0 或 落在下溢区中的数

浮点数范围比定点数大，但数的个数没变多，故数之间更稀疏，且不均匀

三、举例

例 6.13　将 $+\frac{19}{128}$ 写成二进制定点数、浮点数及在定点机和浮点机中的机器数形式。其中数值部分均取 10 位，数符取 1 位，浮点数阶码取 5 位（含1位阶符）。

解：　设 $x = +\frac{19}{128}$

二进制形式　$x = 0.0010011$

定点表示　$x = 0.0010011000$

浮点规格化形式　$x = 0.1001100000 \times 2^{-10}$

定点机中　$[x]_{原} = [x]_{补} = [x]_{反} = 0.0010011000$

浮点机中　$[x]_{原} = 1, 0010;\ 0.\ 1001100000$

$[x]_{补} = 1, 1110;\ 0.\ 1001100000$

$[x]_{反} = 1, 1101;\ 0.\ 1001100000$

练习题： 将 –58 表示成二进制定点数和浮点数， **6.2**
并写出它在定点机和浮点机中的三种机器数及阶码为移码、尾数为补码的形式。其中数值部分均取 10 位，数符取 1 位，浮点数阶码取 5 位（含1位阶符）。

解：　设 $x = -58$

二进制形式　$x = -111010$

定点表示　$x = -0000\,111010$　　右规

浮点规格化形式　$x = -(0.1110100000) \times 2^{110}$

规格化？

定点机中

$[x]_{原} = 1,0000111010$

$[x]_{补} = 1,1111000110$

$[x]_{反} = 1,1111000101$

浮点机中

$[x]_{原} = 0,0110;1.1110100000$

$[x]_{补} = 0,0110;1.0001100000$

$[x]_{反} = 0,0110;1.0001011111$

$[x]_{阶移、尾补} = 1,0110;1.0001100000$

机器零

P230：当浮点数阶码（原码表示时）小于最小阶码时，称为下溢，按机器零处理

6.2

- 当浮点数 尾数为 0 时，不论其阶码为何值按机器零处理

（移码表示时） （原码或补码表示时）

- 当浮点数 阶码等于或小于它所表示的最小数 时，不论尾数为何值，按机器零处理

如 $m = 4$ $n = 10$

当阶码和尾数都用补码表示时，机器零为

×, × × × ×； 0. 0 0 ⋯ 0

见P273 10, × × × ×； ×.×× ⋯ ×

当阶码用移码，尾数用补码表示时，一种机器零为

0, 0 0 0 0； 0. 0 0 ⋯ 0

全零的机器零 有利于机器中“ 判 0 ”电路的实现

浮点数的另一种规格化表示

规定：小数点前总是“1”，故可隐含表示

注意：和前面例子(0.1xxx)的规定不太一样, 显然这里更合理!

- Normal format（规格化数形式）：

  $+/-1.xxxxxxxxxx \times 2^{Exponent}$

- 32-bit 规格化数：

31		0
S	Exponent	Significand
1 bit	? bits	? bits

S 是符号位（Sign）

Exponent 阶码用移码（增码）来表示

Significand (also mantissa or coefficient) 表示 xxxxxxxxxxxxx，尾数部分

(基可以是 2 / 4 / 8 / 16，约定信息，无需显式表示)

- 早期的计算机，各自定义自己的浮点数格式

问题：浮点数表示不统一会带来什么问题？

“Father” of the IEEE 754 standard

直到80年代初，各个机器内部的浮点数表示格式还没有统一
因而相互不兼容，机器之间传送数据时，带来麻烦

1970年代后期， IEEE成立委员会着手制定浮点数标准

1985年完成浮点数标准IEEE 754的制定

现在所有计算机都采用IEEE 754来表示浮点数

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.

Prof. William Kahan

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html
https://amturing.acm.org/award_winners/kahan_1023746.cfm

四、IEEE 754 标准

S	阶码（含阶符）	尾　　数

数符　　小数点位置

尾数为规格化表示 　$+/-\ 1.xxxxxxxxxx_{two} \times 2^{Exponent}$

非 "0" 的有效位最高位为 "1" （隐含）

	符号位 *S*	阶码	尾数 （这里的尾数不含数符）	总位数
短实数	1	8	23 单精度	32
长实数	1	11	52 双精度	64
临时实数	1	15	64 扩展精度	80

IEEE 754 Floating Point Standard

规格化数：+/-1.xxxxxxxxxx$_{two}$ x 2^{Exponent}

Single Precision(单精度)：(Double Precision(双精度)类似)

S	Exponent	Significand
1 bit	8 bits	23 bits

- Sign bit: 1 表示negative；0表示positive
- Exponent（阶码 / 指数）：　全0和全1用来表示特殊值！
  - •SP规格化数阶码范围为0000 0001 (-126) ~ 1111 1110 (127)
    - 1-127 = -126　　254-127 = 127
  - •bias为127 (single), 1023 (double)
    - 为什么用127？若用128，则阶码范围为多少？
    - 0000 0001 (-127) ～ 1111 1110 (126)
- Significand（尾数）：
  - • 规格化尾数最高位总是1，所以隐含表示，省1位
  - • 1 + 23 bits （single），1 + 52 bits （double）

SP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

DP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-1023)}$

在线IEEE-754计算网页：http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.h

C语言中的变量值 vs. 数值在机器中的编码

```c
#include<stdio.h>
int main()
{
    int          a = -1;
    unsigned int b = -1;
    int          c = -20;
    int          d = 20;
    float         e = 0.5f;
    printf ("a = %d;\t\t b = %u;\t c = %d;\t\t d = %d;\t  e = %f\n",a,b,c,d,e);
    printf ("a = 0x%08x;\t b = 0x%08x;\t c = 0x%08x;\t d = 0x%08x;  e = 0x%08x\n",a,b,c,d,*(unsigned int*)(&e));
    return 0;
}
```

Output:

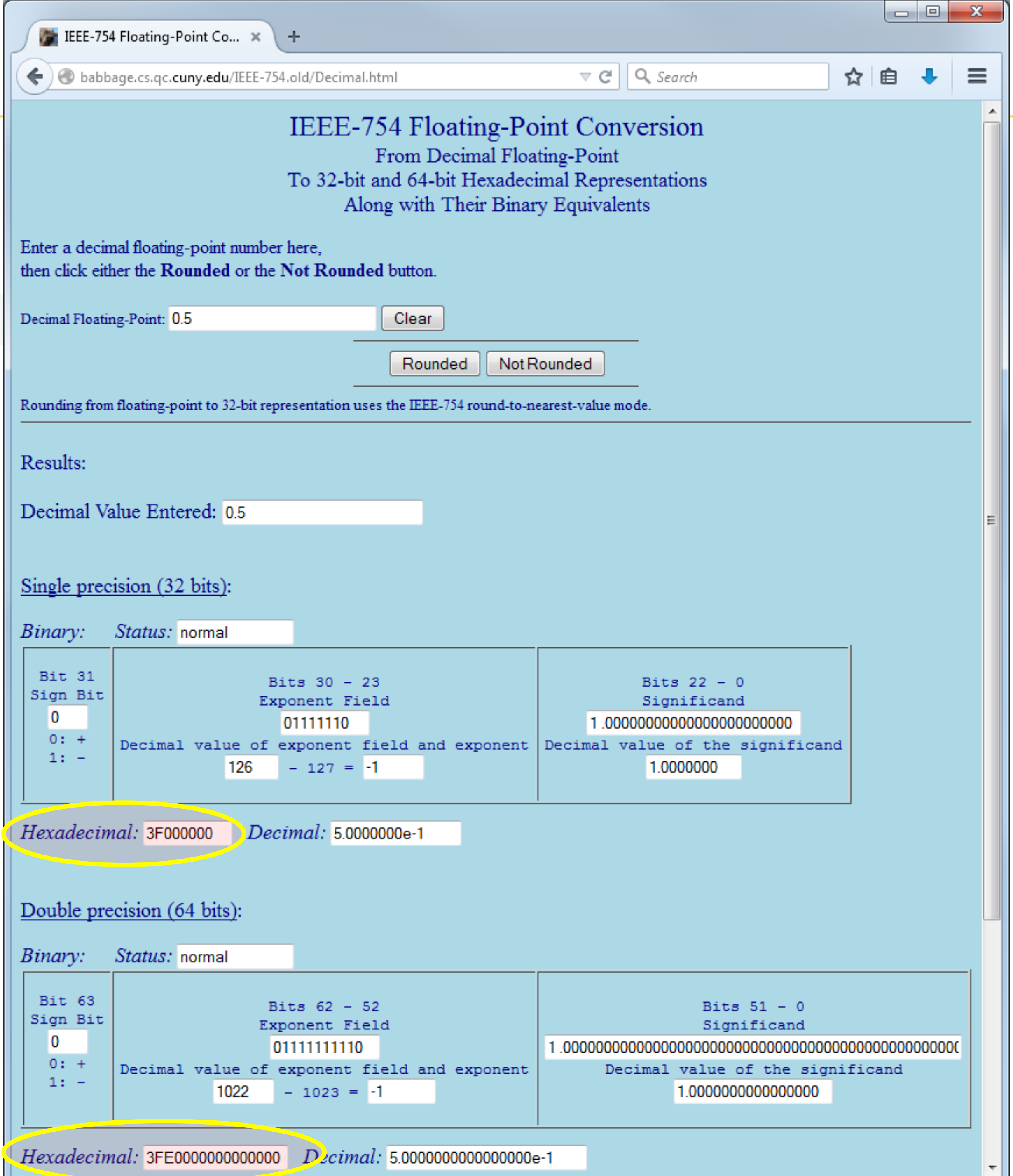
```
a = -1;         b = 4294967295;     c = -20;          d = 20;           e = 0.500000
a = 0xffffffff; b = 0xffffffff;     c = 0xffffffec;   d = 0x00000014;   e = 0x3f000000
```

- ❑ 定点整数的表示 → 补码
- ❑ 浮点数的表示规范 → IEEE 754
  - IEEE Standard for Floating-Point Arithmetic
  - 高性能计算机性能：**FL**OPS

[hw3s]
选做作业

```
float
```

```
e = 0.500000
e = 0x3f000000
```



C语言程序中的整数

无符号数：unsigned int (short / long)；带符号整数： int (short / long)

常在一个数的后面加一个“u”或“U”表示无符号数

假定以下关系表达式在32位用补码表示的机器上执行，结果是什么？

关系表达式
0 == 0U
-1 < 0
-1 < 0U
2147483647 > -2147483647-1
2147483647U > -2147483647-1
2147483647 > (int) 2147483648U
-1 > -2
(unsigned) -1 > -2

[hw3s]

选做作业

C语言程序中的整数(扩展操作)

在32位机器上输出si, usi, i, ui的十进制（真值）和十六进制值（机器数）是什么？

```
short  si = -32768;
unsigned short  usi = si;
int  i = si;
unsigned  ui = usi ;
```

[hw3s]
选做作业

[hw3s]选做作业

- (1) "-1"是个很常见的数值，但其补码用了那么多bit表示。如果是数据的传输和压缩存储时很浪费。你能想到有什么方法可以避免这样的情况吗？换句话，常见的数值能否用短一些的编码表示和传输？
- (2)以下关系表达式在32位（64位亦可）用补码表示的机器上执行，结果是什么？并请分析和解释原因。（如果使用代码进行验证，请说明机器环境是32位还是64位）

```
0 == 0U
-1 < 0
-1 < 0U
2147483647 > -2147483647-1
2147483647U > -2147483647-1
2147483647 > (int) 2147483648U
-1 > -2
(unsigned) -1 > -2
```

- (3) 在32位（64位亦可）机器上输出si, usi, i, ui的十进制（真值）和十六进制值（机器数）是什么？并请分析和解释原因。

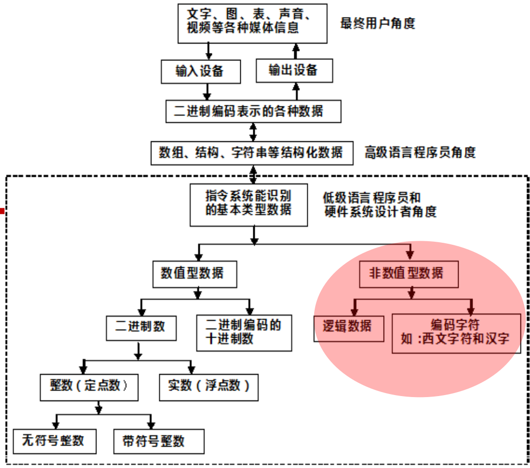
```
short  si = -32768;
unsigned short  usi = si;
int  i = si;
unsigned  ui = usi ;
```

- 本次作业提交截止时间：
  - 2023年3月20日上课前提交

十进制数(Decimal)的编码表示

- **计算机中为什么要用十进制数表示数值？**
  - 日常使用的都是十进制数，所以，计算机外部都使用十进制数
  - 在一些有大量数据输入/出的系统中，为减少二进制数和十进制数之间的转换，在计算机内部直接用十进制数表示数值
- **ASCII码**
  - American Standard Code for Information Interchange，美国信息交换标准码
  - 7位二进制码，表示128种字符
  - 可表示十进制数串，非数值计算使用
- **BCD码**
  - Binary Coded Decimal
  - 用4位二进制代码表示一位十进制数，可进行算术运算
- **详见第5章附录5A和5B**

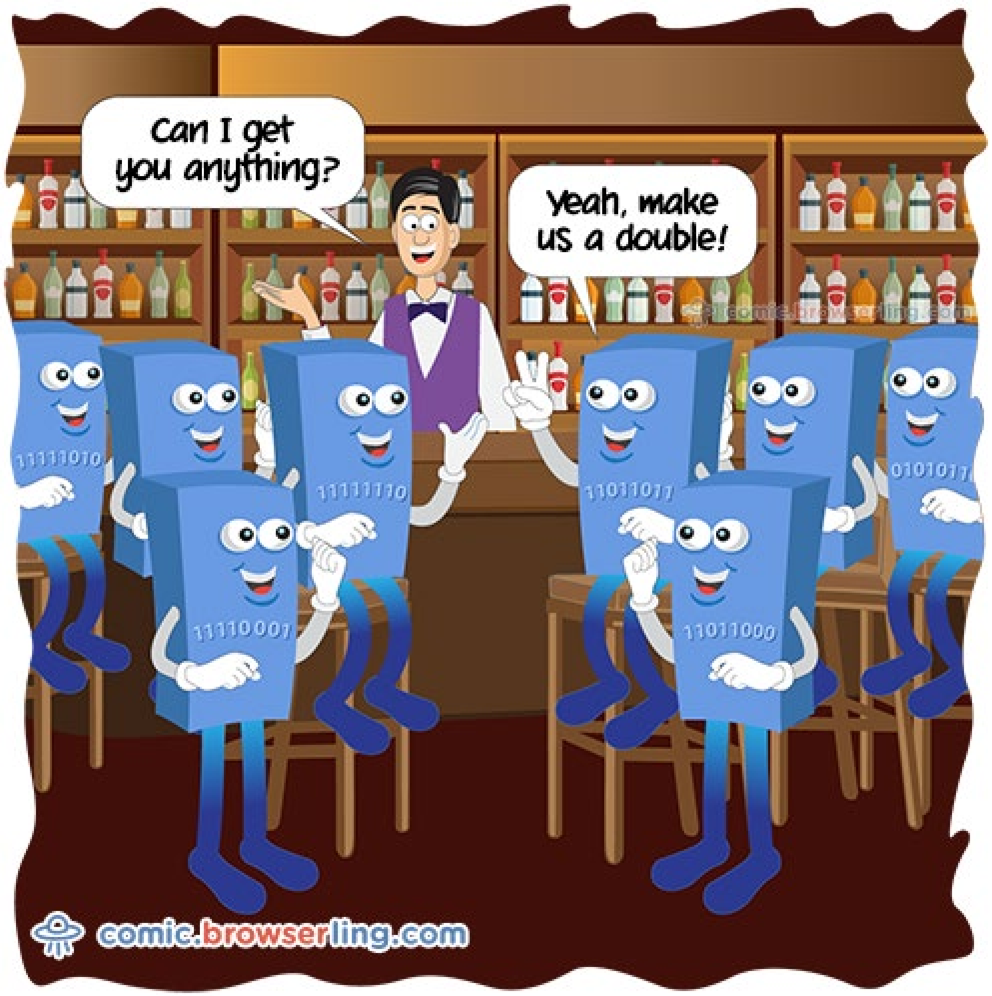
非数值数据的表示与编码



- **逻辑数据：真1 / 假0 （例如处理器中的标志寄存器）**
- **西文字符：ASCII码**
- **汉字及国际字符编码**
  - 汉字国际码GB2312-80，1981年《信息交换用汉字编码字符集·基本集》
  - 国际多字符集
    - 为使所有国际字符都能互换，必须创建一种涵盖全部字符的多字符集，避免出现乱码
    - Windows操作系统(中文版)已采用中西文统一编码，收集了中、日、韩三国常用的约2万汉字，称为“Unicode”，采用2字节编码，与UCS-2一致
    - https://www.freecodecamp.org/news/everything-you-need-to-know-about-encoding/
- **机器级数据的检错和纠错（第4章“存储器”相关章节）**
  - 奇偶校验码(Parity)：适应于一字节长数据的校验（不能纠错），如内存
  - ECC（Error Checking and Correcting）：用于错误的检查并可纠正错误
  - 汉(海)明校验码：各组内用奇偶校验，用于内存储器数据的校验及纠错
  - 循环冗余校验码(CRC)：用于通信和外存，适合于大批量数据校验及纠错
- **数据在存储器中的字节存放次序，以及在传输通路中的字节传输次序**
  - 数据串的MSB放在低地址 vs. 高地址
  - 大端(Big Endian) vs. 小端(Little Endian)
  - 唐朔飞教材P306-图7.4

http://www.ict.cas.cn/xwgg/jssxw/201908/t20190812_5358477.html

Eight bytes walk into a bar



作业

- 【hw3】
- 课后习题：6.10，6.15，6.16

- 本次作业提交截止时间：
  - **2023年3月20日上课前**提交