

《计算机系统基础》习题课

第二章 数据的机器级表示与处理

2019年9月25日

课后作业习题

- 第二版课本第2章习题， pg. 79开始， 习题号：
 - 3、 9、 10、 15、 21、 29、 33、 40
- 课后作业习题Deadline
 - 2019年10月8日24点， 即， 2019年10月9日0点
- 作业以电子稿形式上传到cms系统

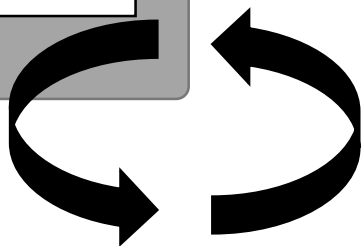
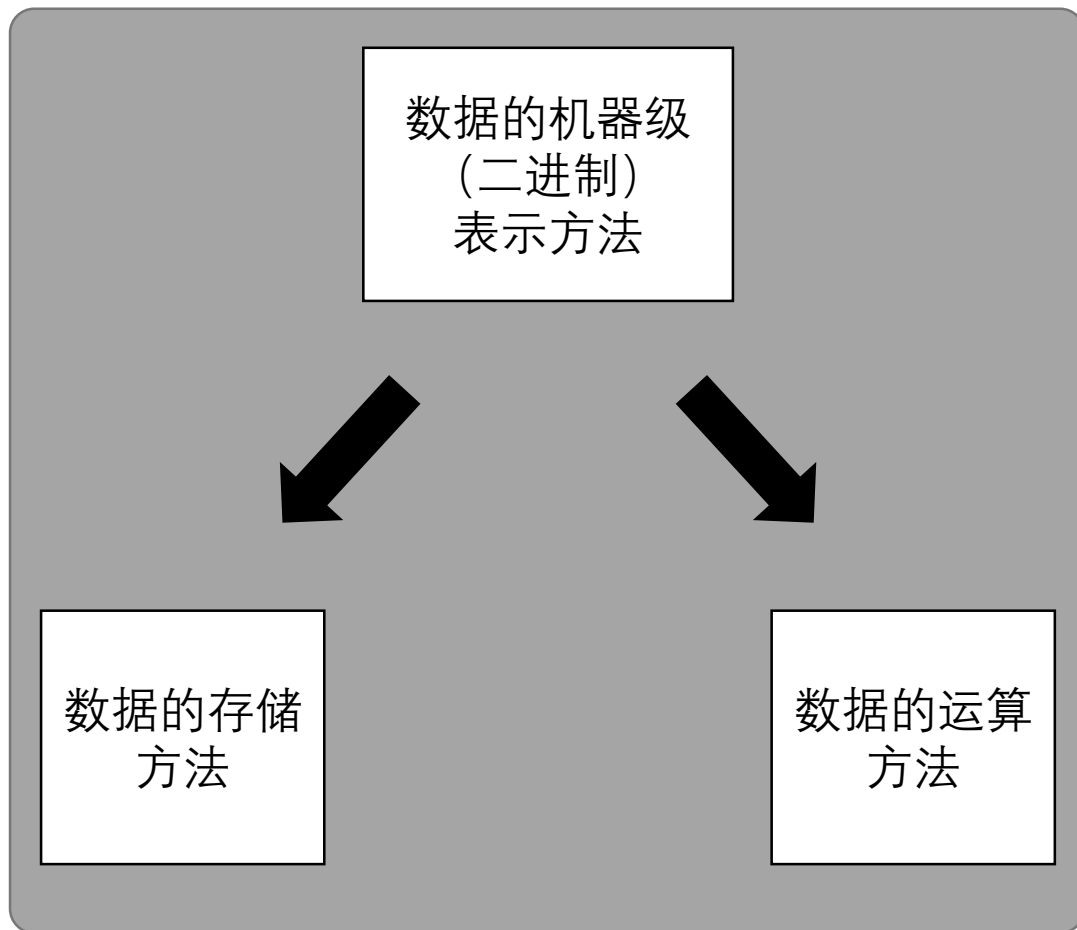
本章内容总览（三个侧面）

1 数据的类型



X

2 对应的问题



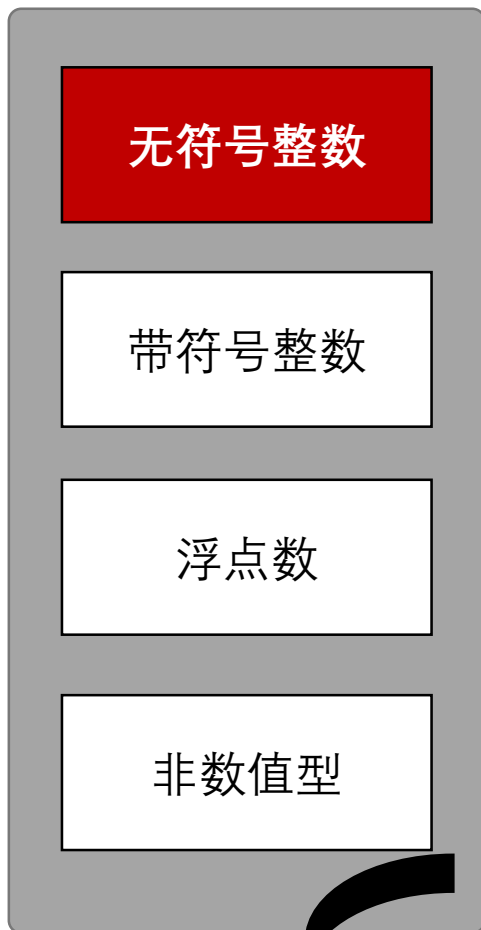
3 数据类型间的转换

无符号整数

unsigned short, unsigned int, uint8_t, uint16_t, uint32_t, ...

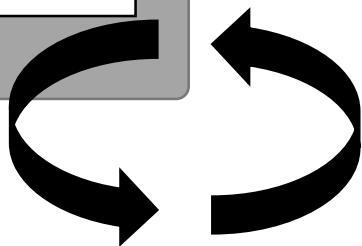
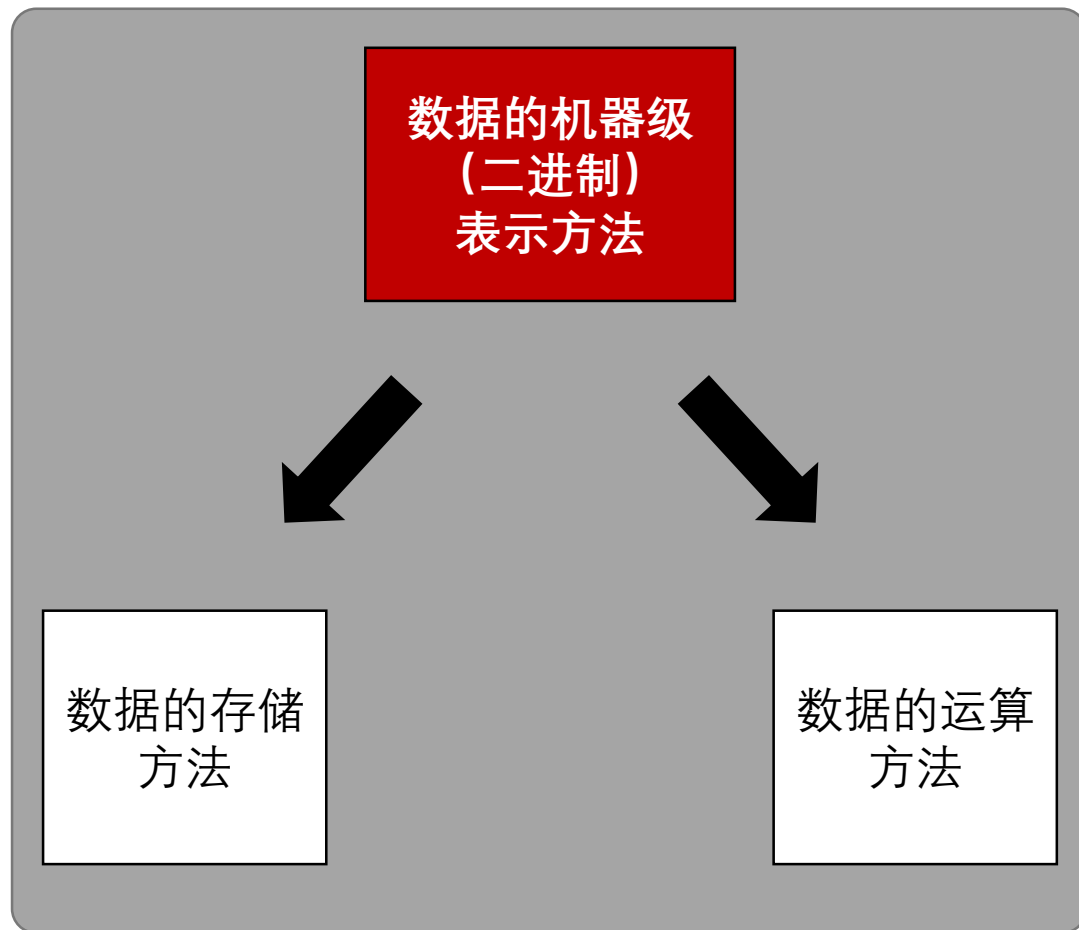
内容1.1：无符号整数的机器级表示

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

内容1.1：无符号整数的机器级表示

- 二进制编码

世界上只有10种人，懂二进制的和不懂二进制的

- 要点：不同进制之间的转换方法，熟记

例：
1010B
= 12O
= 10D
= AH


内容1.1：无符号整数的机器级表示

- 二进制编码

世界上只有10种人，懂二进制的和不懂二进制的

- 要点：不同进制之间的转换方法，熟记

例：

	1010	B		3个二进制位构成1个八进制位
=	12	O		
=	10	D		
=	A	H		

内容1.1：无符号整数的机器级表示


- 二进制编码

世界上只有10种人，懂二进制的和不懂二进制的

- 要点：不同进制之间的转换方法，熟记

例：

	1010	B	
=	12	O	
=	10	D	
=	A	H	



4个二进制位构成1个十六进制位

内容1.1：无符号整数的机器级表示

- 二进制编码

世界上只有10种人，懂二进制的和不懂二进制的

- 要点：不同进制之间的转换方法，熟记

例：

1010B
=
12O
=
10D
=
AH

二进制换算十进制：二的次方相加

十进制换算二进制：除以2取余法
(记住4, 8, 16, ...等特殊值很有用)，
扩展到定点小数，课本pg. 33

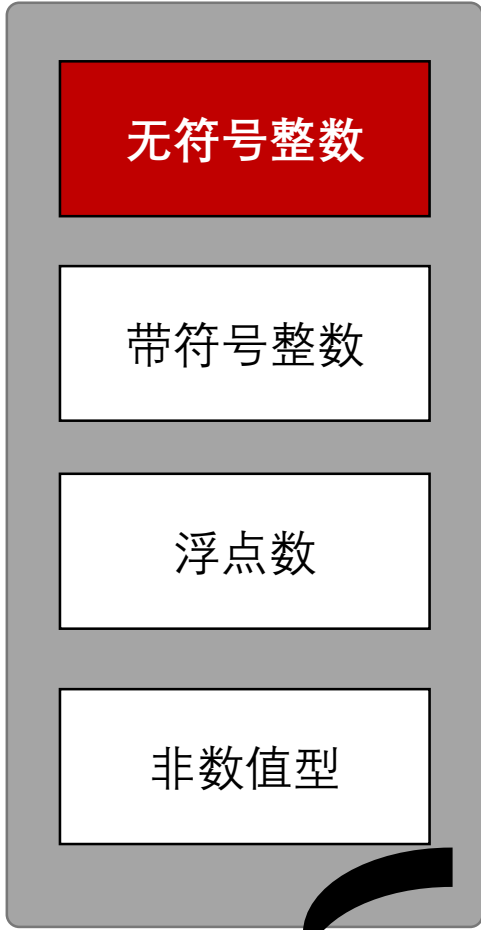
内容1.1：无符号整数的机器级表示

- 课堂习题：

略

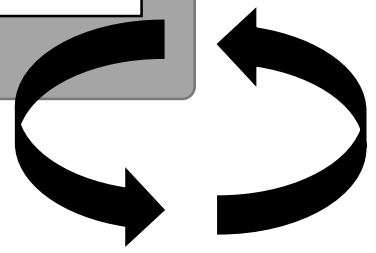
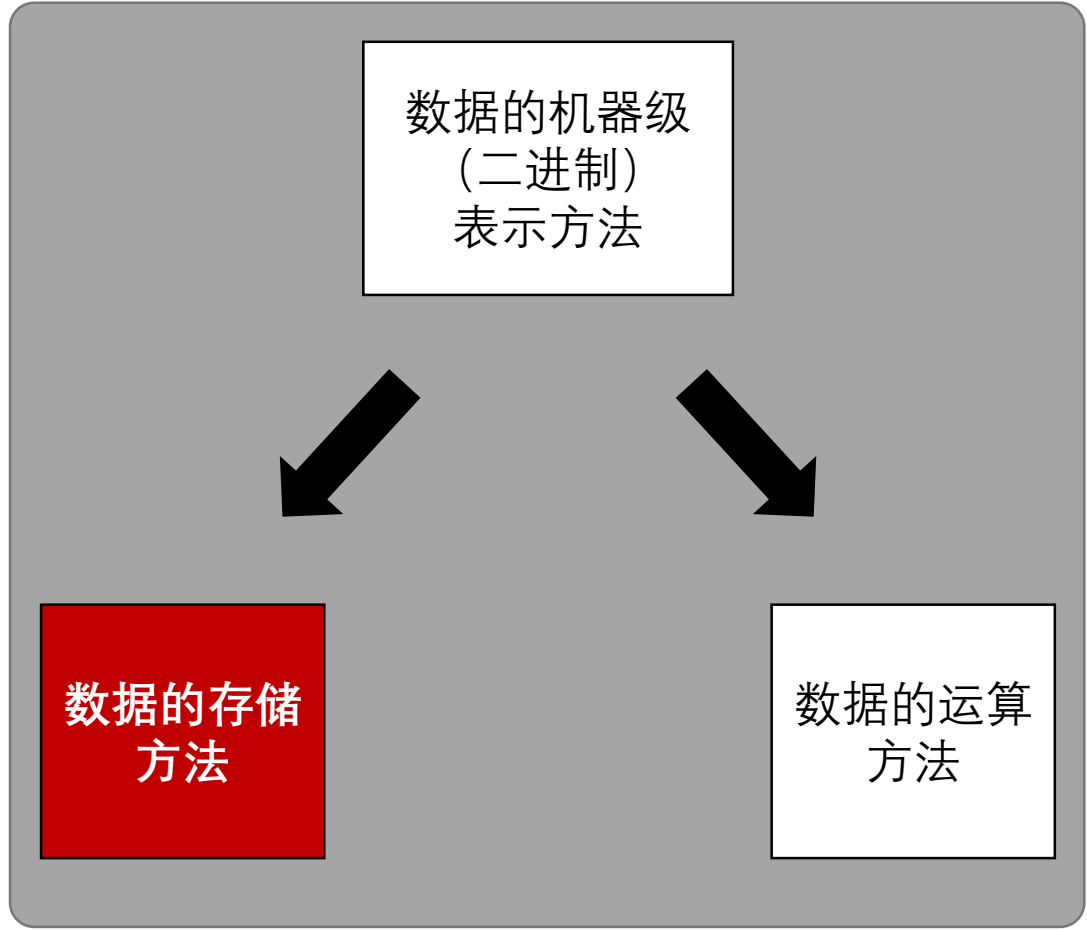
内容1.2：无符号整数的存储方法

1 数据的类型



X

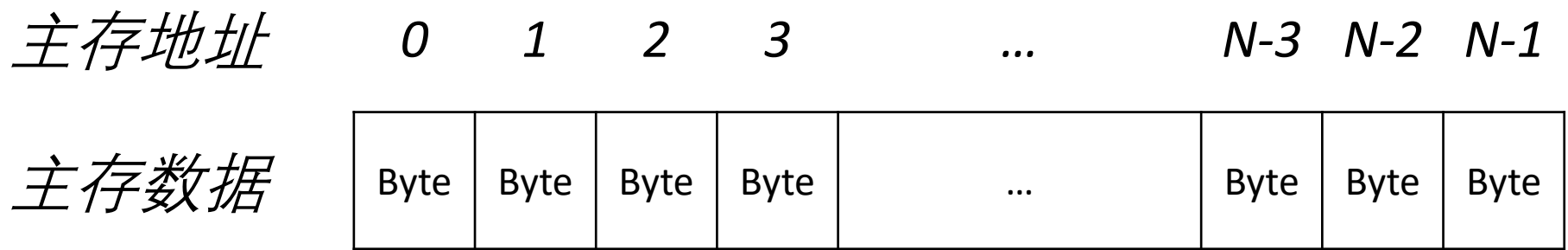
2 对应的问题



3 数据类型间的转换

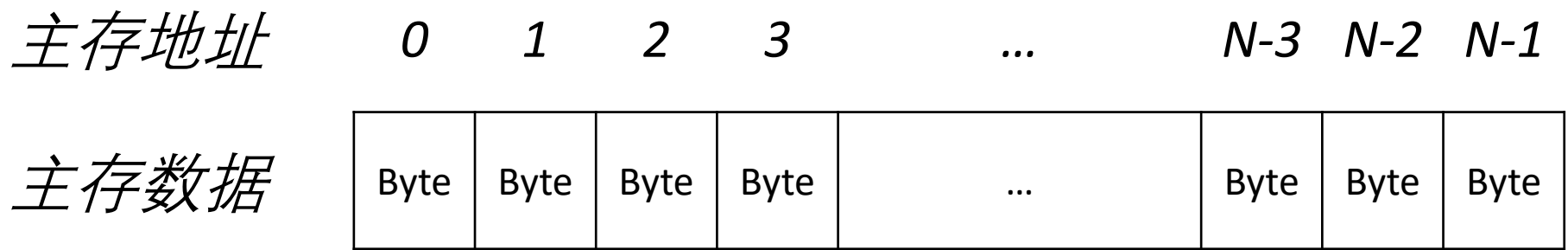
内容1.2：无符号整数的存储方法

- 主存（内存）模型



内容1.2：无符号整数的存储方法

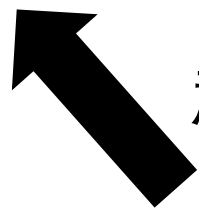
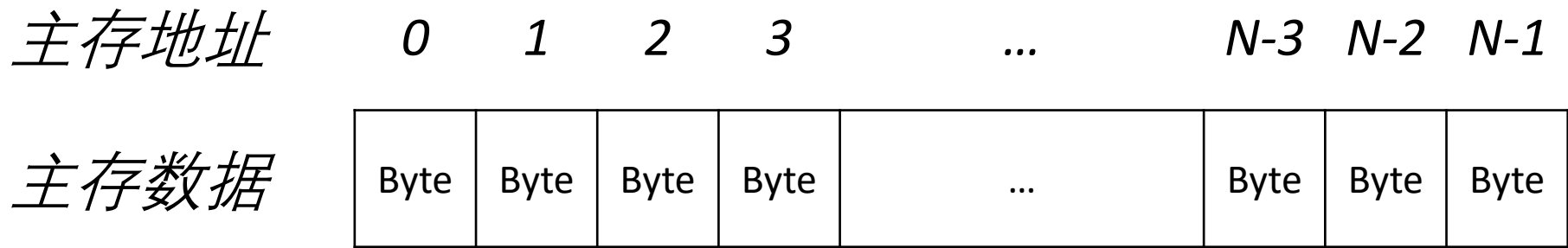
- 主存（内存）模型



NEMU中 N == MEM_SIZE_B == 128M

内容1.2：无符号整数的存储方法

- 主存（内存）模型

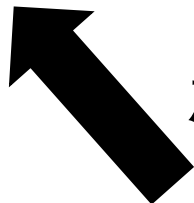
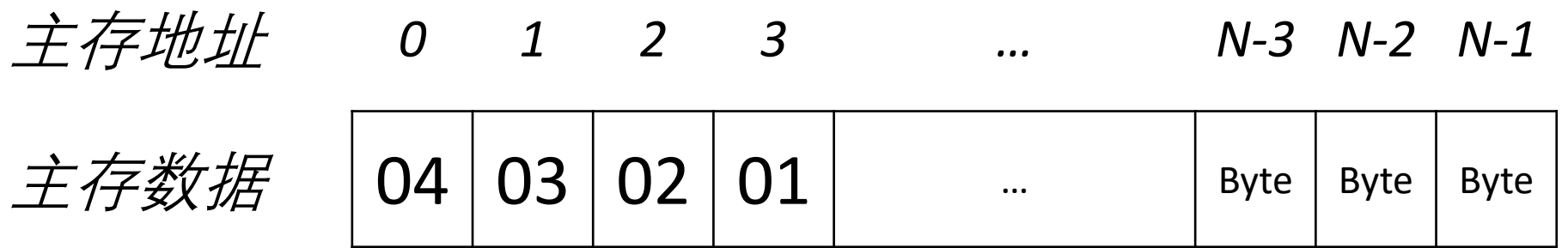


起始地址为0，怎么存？

小端方式： **0x** 01 02 03 04

内容1.2：无符号整数的存储方法

- 主存（内存）模型

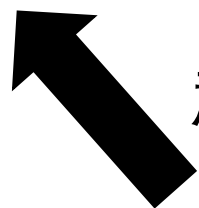
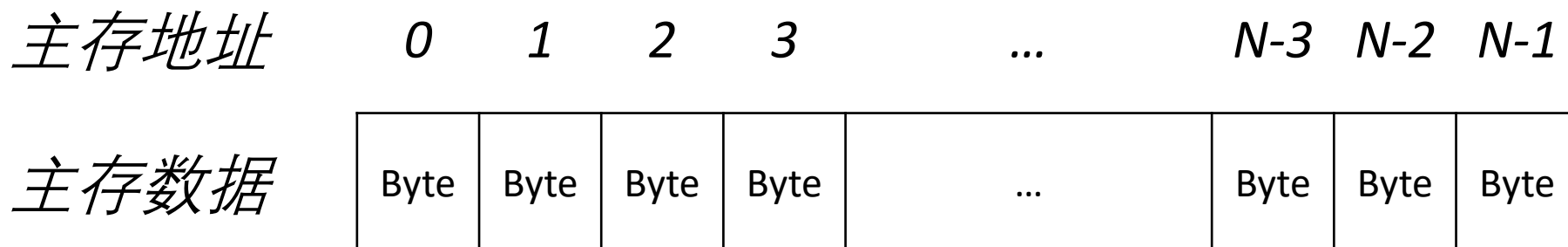


起始地址为0，低有效字节在低地址

小端方式： 0x 01 02 03 04

内容1.2：无符号整数的存储方法

- 主存（内存）模型

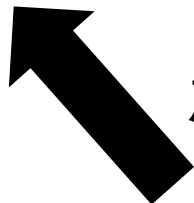
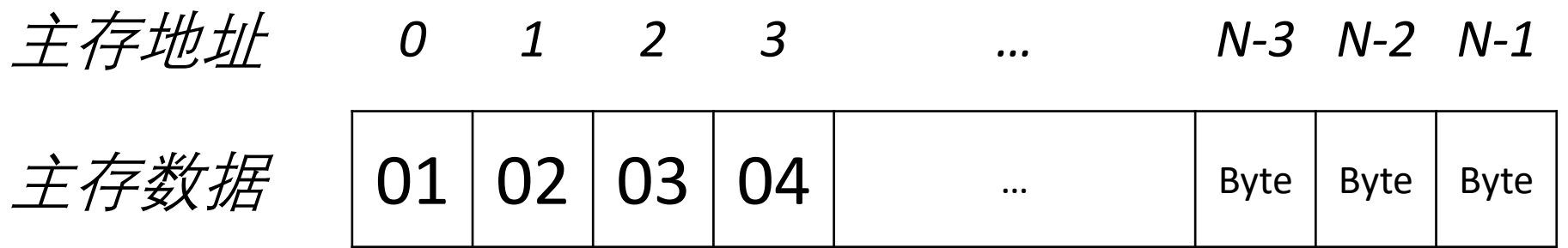


起始地址为0，怎么存？

大端方式： 0x 01 02 03 04

内容1.2：无符号整数的存储方法

- 主存（内存）模型



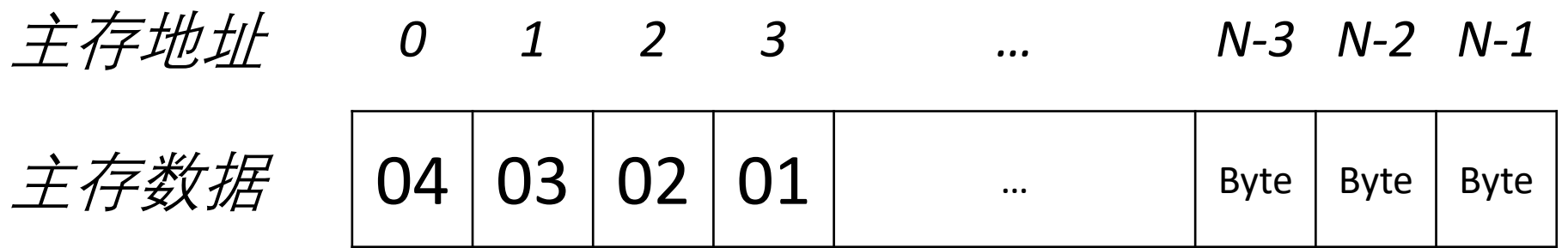
起始地址为0，高有效字节在低地址

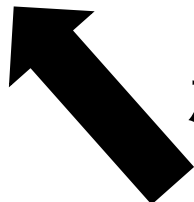
大端方式： 0x 01 02 03 04

内容1.2：无符号整数的存储方法

- 主存（内存）模型

X86小端机！ 大端机少见！



 起始地址为0，低有效字节在低地址

小端方式： 0x 01 02 03 04

内容1.2：无符号整数的存储方法

- 课堂习题1.2.1:

```
uint8_t a[3] = {0x1, 0x2, 0x3};
```

```
unsigned short b[3] = {0x1, 0x2, 0x3};
```

```
unsigned int c[3] = {0x1, 0x2, 0x3};
```

假设起始地址为0，小端机，怎么存？

预习课本pg. 128, 3.5.1节

内容1.2：无符号整数的存储方法

- 课堂习题1.2.2:

```
typedef struct {  
    uint8_t  a;  // 假设 a = 0x1  
    uint16_t b;  // 假设 b = 0x2  
    uint32_t c;  // 假设 c = 0x3  
} SomeStruct;
```

假设起始地址为0，小端机，怎么存？

不考虑对齐，预习课本pg. 132，3.5.2节

考虑对齐，预习课本pg. 137，3.5.4节

内容1.2：无符号整数的存储方法

- 课堂习题1.2.3:

```
typedef union { // 假设初始值为全0
    uint8_t  a;  // 假设执行a = 0x1会怎样?
    uint16_t b;  // 假设执行b = 0x2会怎样?
    uint32_t c;  // 假设执行c = 0x3会怎样?
} SomeUnion;
```

假设起始地址为0，小端机，怎么存？执行上述注释内容的语句会怎样？

预习课本pg. 135， 3.5.3节

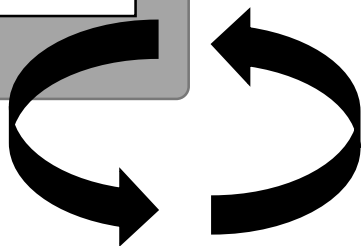
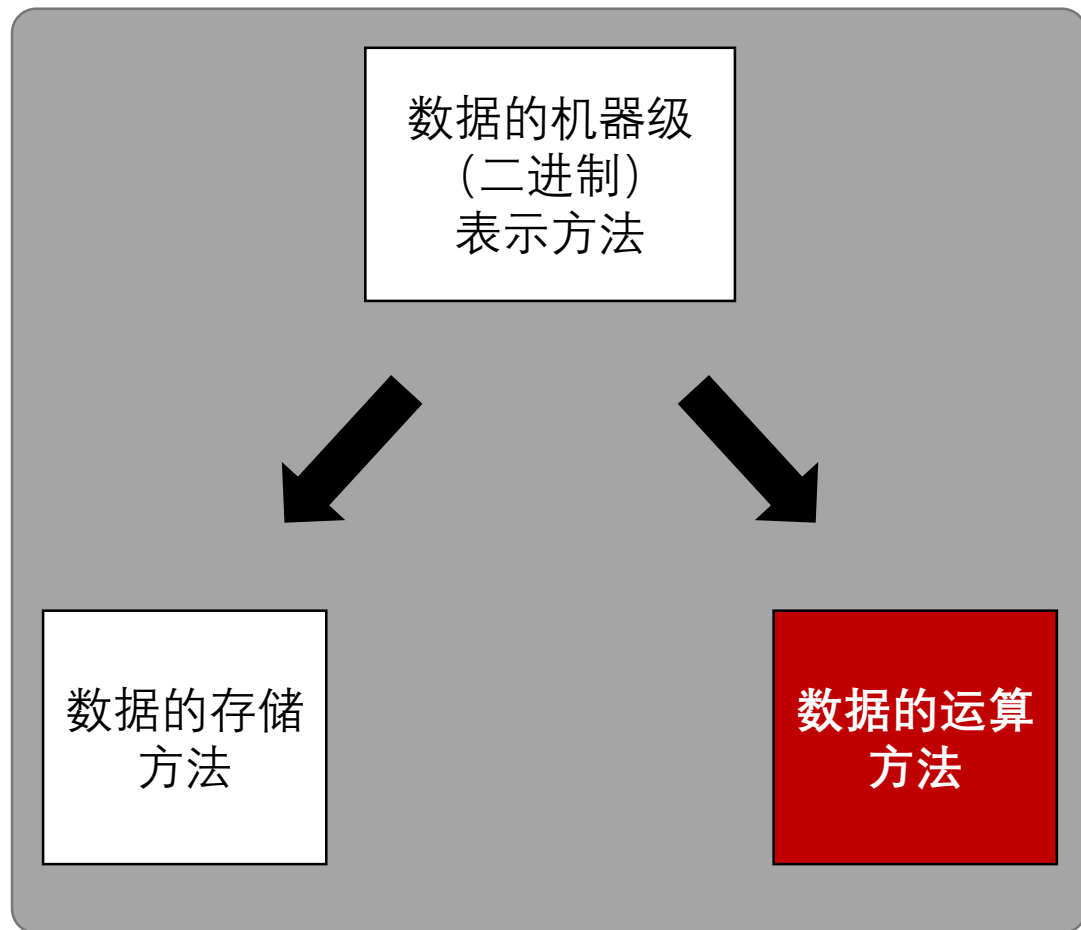
内容1.3：无符号整数的运算方法

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

内容1.3：无符号整数的运算方法

- 数学运算
 - 加减、乘除
- 位运算
 - 逻辑和算数移位
 - 按位或、按位与
 - 位扩展（带符号扩展，无符号扩展）、位截断
- 逻辑运算
 - C语言中0为False，1为True
 - 与、或、非

内容1.3：无符号整数的运算方法

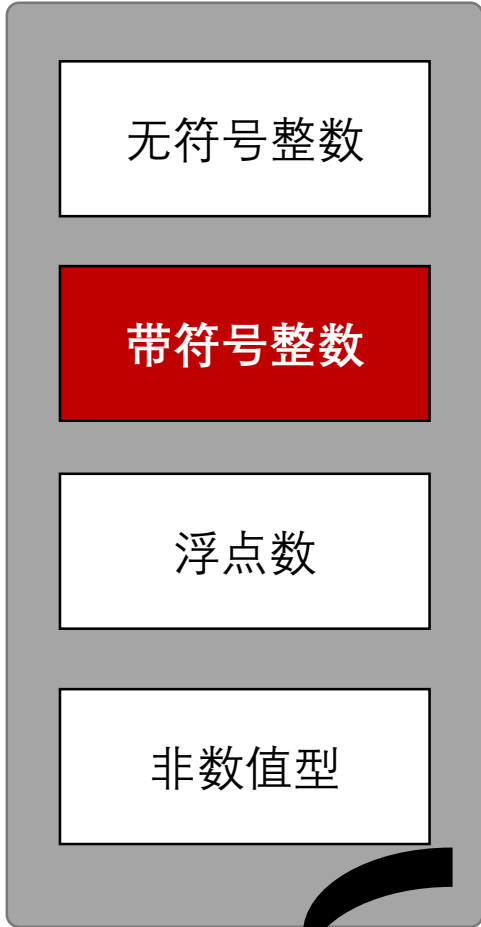
- 要点：
 - 加法和减法的统一（涉及补码表示法）
 - 标志位ZF, OF, SF, CF
 - 结果的溢出及其处理方法
- 课堂习题：略，做PA 1.2

带符号整数

short, int, long, ...

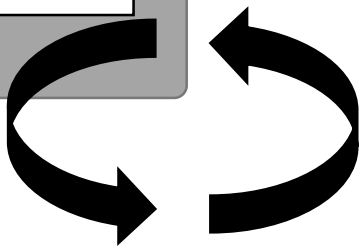
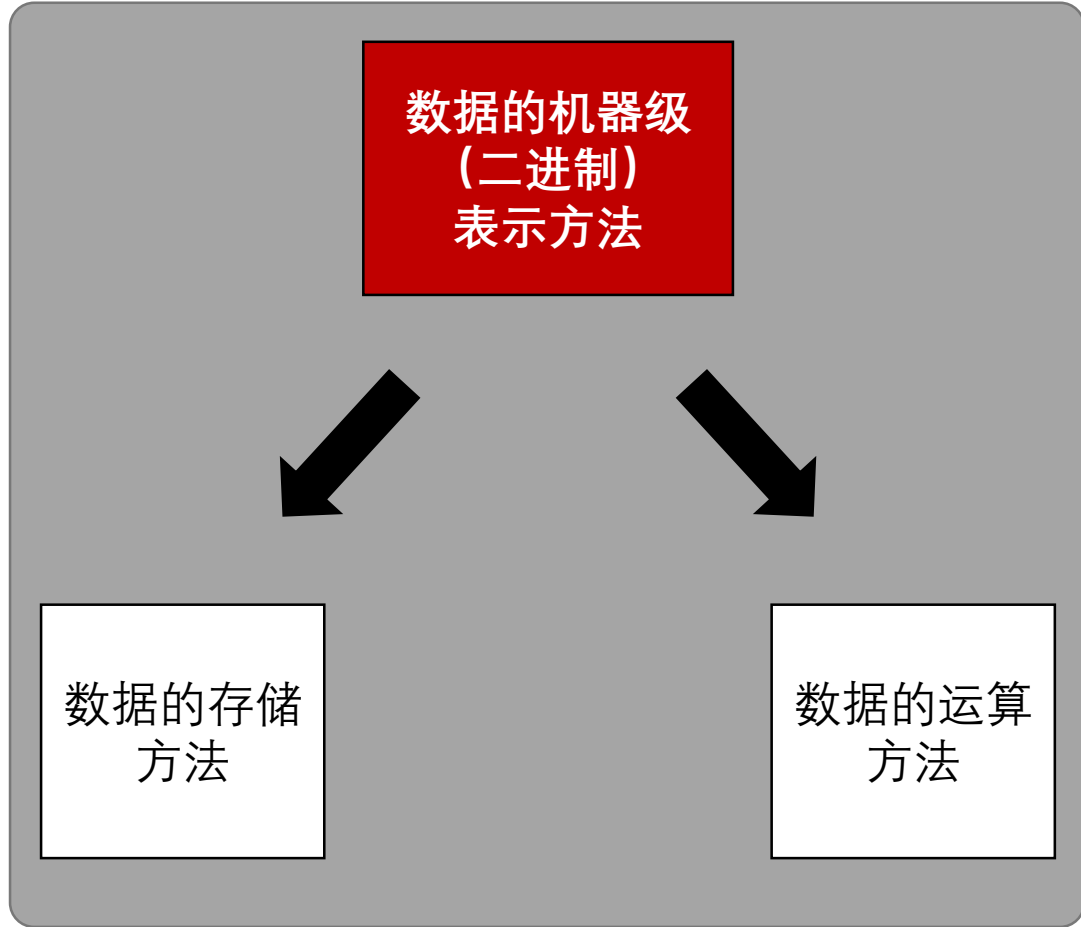
内容2.1：带符号整数的机器级表示

1 数据的类型



X

2 对应的问题



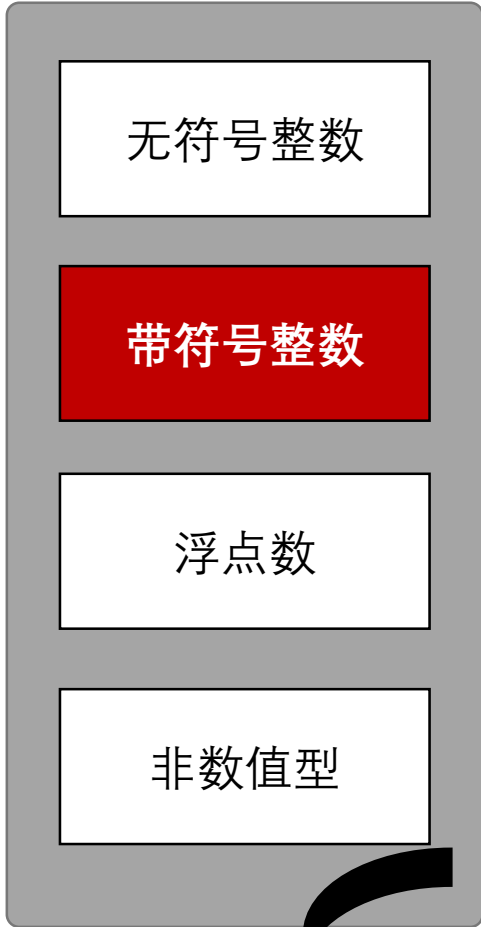
3 数据类型间的转换

内容2.1：带符号整数的机器级表示

- 搞清楚几个概念：
 - 真值：1, -1, 0, 17, -17, ...
 - 机器数：真值对应机器内部的01编码
- 带符号数的机器级表示方法
 - 原码 – 整数不常用，浮点数采用
 - 反码 – 少见
 - 补码 – 此小节重点，十分简单

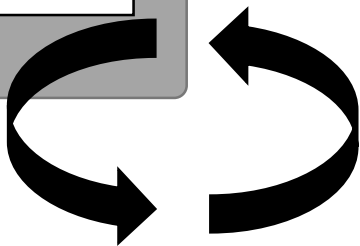
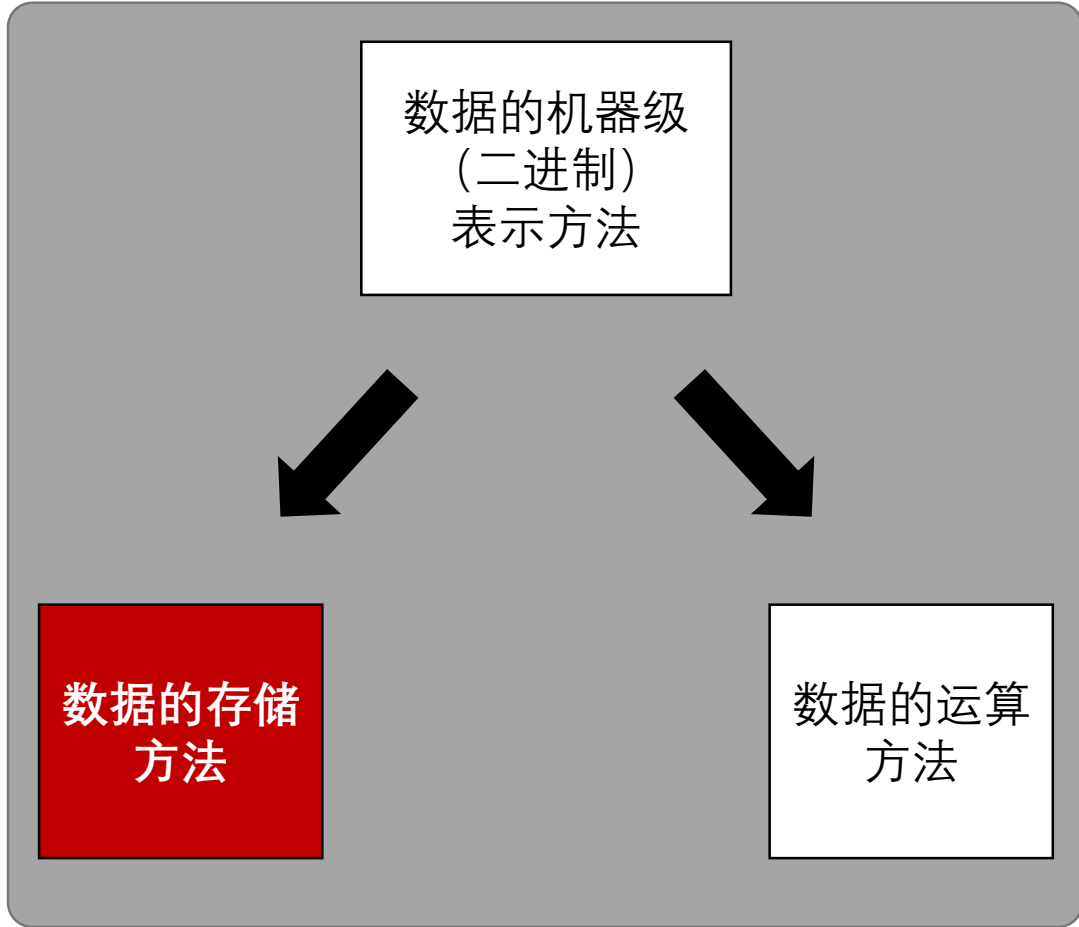
内容2.2：带符号整数的存储方法

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

内容2.2：带符号整数的存储方法

- 不管是什么类型的数据，在考虑其存储方法时
 - 首先将其转换成对应的机器级表示
 - 将机器级表示的数据当成无符号数，套用内容1.2所述的存储方法
- 上述方法适用于
 - 无符号数
 - 带符号数
 - 浮点数
 - 非数值型数据
 - 所有一切可以编码的数据（若无特殊例外说明）
- 所以存储方法这个话题以后就不讲了

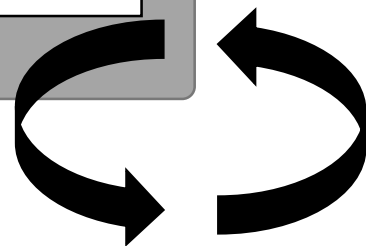
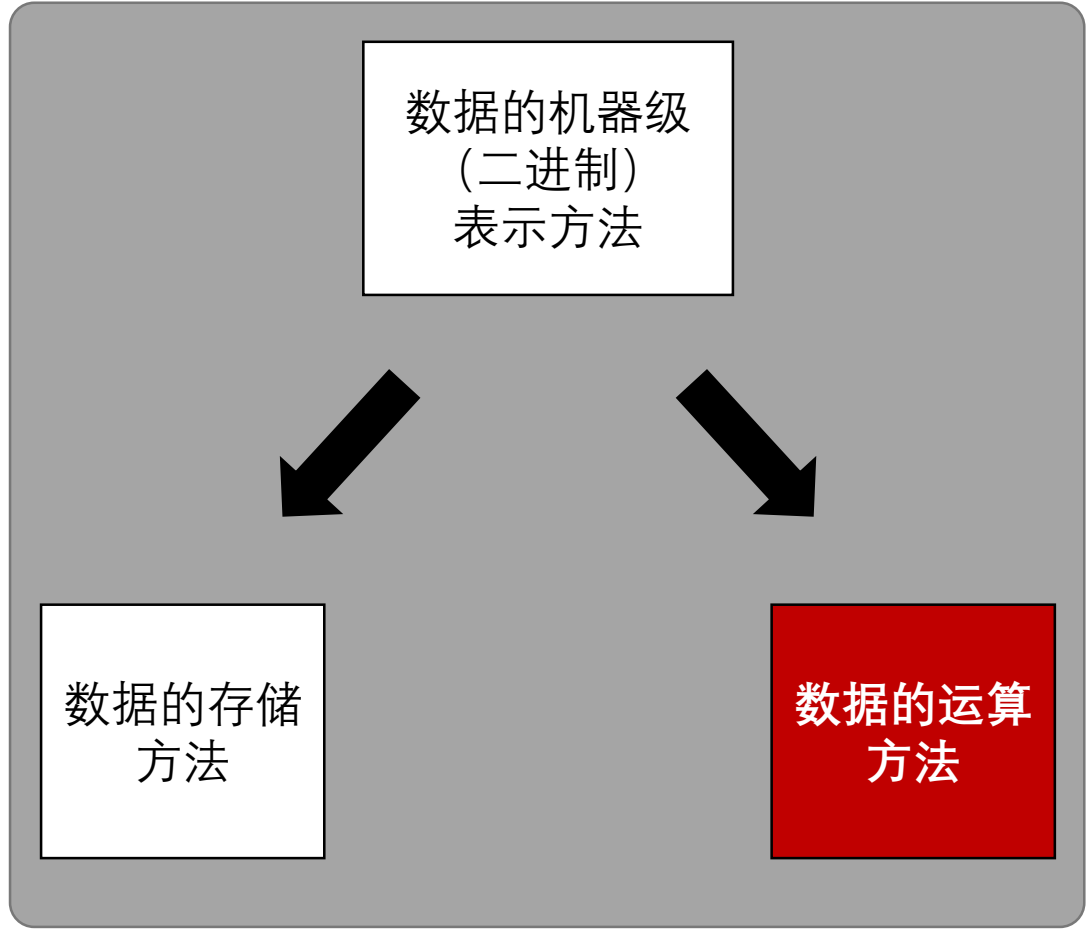
内容2.3：带符号整数的运算方法

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

内容2.3：带符号整数的运算方法

- 一言以蔽之

补码搞定一切, 统一无符号带符号整数的加减法, *OF*的解释注意一下

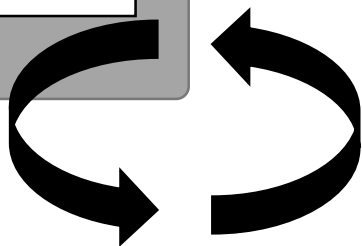
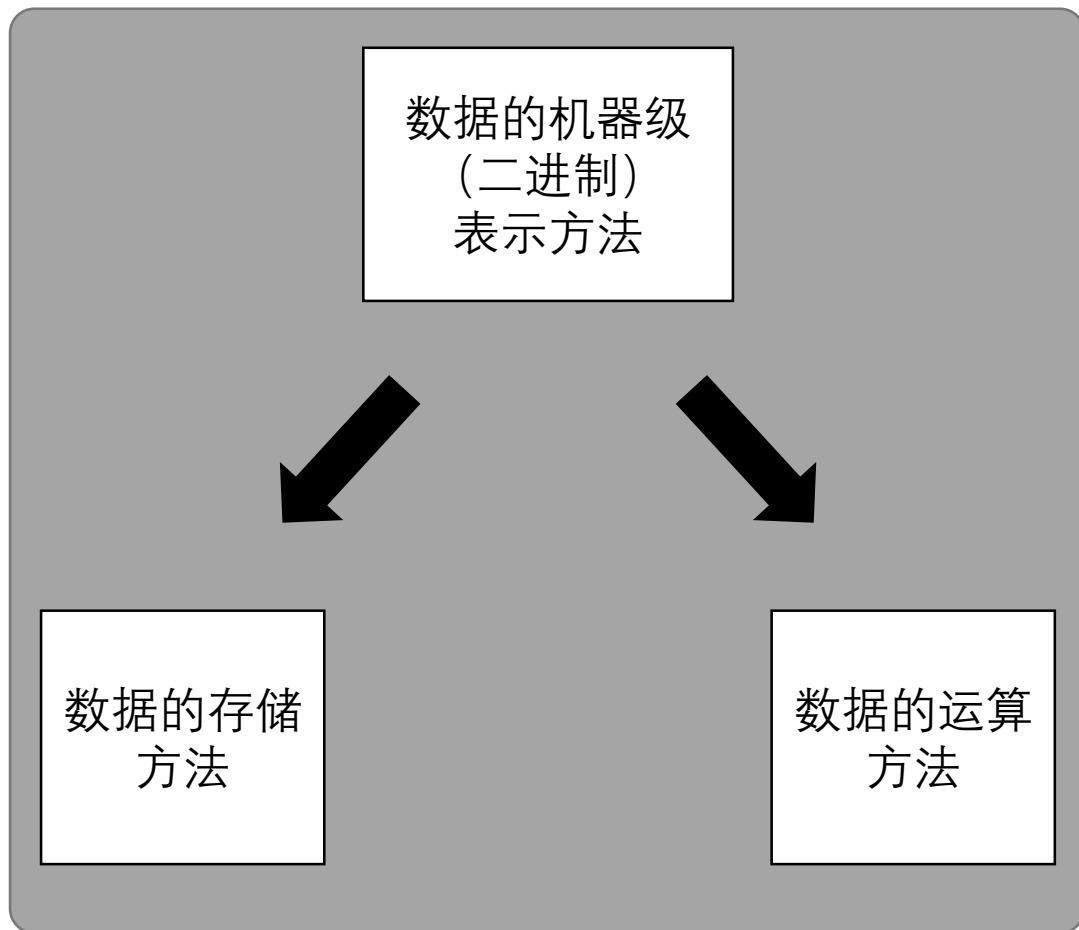
内容2.4：无符号和带符号整数的转换

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

内容2.4：无符号和带符号整数的转换

- 表示范围，课题习题2.4.1：

16位无符号整数的表示范围是多少？

16位带符号整数的表示范围是多少？

内容2.4：无符号和带符号整数的转换

- 表示范围，课题习题2.4.1：

16位无符号整数的表示范围是多少？

$$0 \sim (2^{16}-1)$$

16位带符号整数的表示范围是多少？

$$-2^{15} \sim (2^{15}-1)$$

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.2
 - 整数的位扩展与位截断（同等长度的数据）

```
short s = 0;
unsigned short us = 0;
int i = 0;
unsigned int ui = 0;

s = -32768;
us = s;
printf("us = %d, 0x%04x\n", us, us);

us = 32768;
s = us;
printf("s = %d, 0x%04x\n", s, s);
```

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.2

- 整数的位扩展与位截断（同等长度的数据）

```
short s = 0;  
unsigned short us = 0;  
int i = 0;  
unsigned int ui = 0;
```

宽度相同的无符号数和带符号数之间的转换
原则是保持机器表示不变（解释不同）。

```
s = -32768;  
us = s;  
printf("us = %d, 0x%04x\n", us, us);
```

us = 32768, 0x8000

```
us = 32768;  
s = us;  
printf("s = %d, 0x%04x\n", s, s);
```

s = -32768, 0xffff8000

内容2.4：无符号和带符号整数的转换

• 课堂习题2.4.2

```
s = -32768;
```

```
i = s;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

```
us = -32768;
```

```
i = us;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

```
s = -32768;
```

```
ui = s;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

```
us = -32768;
```

```
ui = us;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

- 整数的位扩展与位截断（位扩展）

内容2.4：无符号和带符号整数的转换

• 课堂习题2.4.2

- 整数的位扩展与位截断（位扩展）

```
s = -32768;
```

```
i = s;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

```
i = -32768, 0xffff8000
```

```
us = -32768;
```

```
i = us;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

```
i = 32768, 0x00008000
```

```
s = -32768;
```

```
ui = s;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

```
ui = -32768, 0xffff8000
```

```
us = -32768;
```

```
ui = us;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

```
ui = 32768, 0x00008000
```

内容2.4：无符号和带符号整数的转换

• 课堂习题2.4.2

```
s = -32768;
```

```
i = s;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

- 整数的位扩展与位截断（位扩展）

```
us = -32768;
```

```
i = us;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

0 扩展用于无符号数，在短时无符号数前面添加足够的0

i = 32768, 0x00008000

```
s = -32768;
```

```
ui = s;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

```
us = -32768;
```

```
ui = us;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

ui = 32768, 0x00008000

内容2.4：无符号和带符号整数的转换

• 课堂习题2.4.2

```
s = -32768;
```

```
i = s;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

i = -32768, 0xffff8000

```
us = -32768;
```

```
i = us;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

```
s = -32768;
```

```
ui = s;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

ui = -32768, 0xffff8000

```
us = -32768;
```

```
ui = us;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

符号扩展用于补码表示的带符号整数，在短的带符号整数前添加足够多的符号位。

内容2.4：无符号和带符号整数的转换

• 课堂习题2.4.2

```
s = -32768;
```

```
i = s;
```

```
printf("i = %d, 0x%08x\n", i, i);
```

- 整数的位扩展与位截断（位扩展）

```
us = -32768;
```

```
i = us;
```

```
printf("i = %d, 0x%08x\n",
```

Relative order of conversion between data
size and signed/unsigned:

First change the size and then convert
between signed/unsigned

```
s = -32768;
```

```
ui = s;
```

```
printf("ui = %d, 0x%08x\n", ui, ui); ui = -32768, 0xffff8000
```

```
us = -32768;
```

```
ui = us;
```

```
printf("ui = %d, 0x%08x\n", ui, ui);
```

经验：同时变长度和
无符号/带符号的类型
转换的赋值操作别做！
宁肯分两步走！

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.2
 - 整数的位扩展与位截断（位截断）

```
i = 32768;  
s = i;  
printf("s = %d, 0x%04x\n", s, s);
```

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.2
 - 整数的位扩展与位截断（位截断）

```
i = 32768;
```

```
s = i;
```

```
printf("s = %d, 0x%04x\n", s, s);
```

位截断发生在将长数转换为短数时。当将一个有 w 位的长数截断为 k 位的短数时，就是丢弃高 $w-k$ 位。截断操作会产生溢出。

$s = -32768, 0xffff8000$

经验：无符号数的截断很好用，带符号数的截断控制不住，千万别干！

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.3, C语言中的类型转换

课本pg. 41, 2.2.2 节, 例题2.21

判断下列表达式的真假

$-2147483647-1 == 2147483648$

$-2147483647-1 < 2147483647$

$-2147483647-1U < 2147483647$

$-2147483647-1 < -2147483647$

$-2147483647-1U < -2147483647$

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.3, C语言中的类型转换

课本pg. 41, 2.2.2 节, 例题2.21

判断下列表达式的真假

传说中的结果

$-2147483647 - 1 == 2147483648$

true

$-2147483647 - 1 < 2147483647$

true

$-2147483647 - 1U < 2147483647$

false

$-2147483647 - 1 < -2147483647$

true

$-2147483647 - 1U < -2147483647$

true

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.3, C语言中的类型转换

课本pg. 41, 2.2.2 节, 例题2.21

凡事不如试一试

gcc -o main main.c

```
#include <stdio.h>
```

```
int main() {
```

```
    printf("-2147483647-1 == 2147483648, %d\n", -2147483647-1 == 2147483648);
```

```
    printf("-2147483647-1 < 2147483647, %d\n", -2147483647-1 < 2147483647);
```

```
    printf("-2147483647-1U < 2147483647, %d\n", -2147483647-1U < 2147483647);
```

```
    printf("-2147483647-1 < -2147483647, %d\n", -2147483647-1 < -2147483647);
```

```
    printf("-2147483647-1U < -2147483647, %d\n", -2147483647-1U < -2147483647);
```

```
    return 0;
```

```
}
```

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.3, C语言中的类型转换

课本pg. 41, 2.2.2 节, 例题2.21

凡事不如试一试

gcc -o main main.c

./main的运行结果：

-2147483647-1 == 2147483648, 0

答案错了么！

-2147483647-1 < 2147483647, 1

-2147483647-1U < 2147483647, 0

-2147483647-1 < -2147483647, 1

-2147483647-1U < -2147483647, 1

再仔细看看例题2.21，似乎和C90还是C99有关

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.3, C语言中的类型转换

课本pg. 41, 2.2.2 节, 例题2.21

凡事不如试一试

gcc -std=c90 -o main main.c

./main的运行结果：

main.c: In function 'main':

抛了个warning

main.c:4:2: warning: this decimal constant is unsigned only in ISO C90

```
printf("-2147483647-1 == 2147483648, %d\n", -2147483647-1 == 2147483648);  
^~~~~~
```

-2147483647-1 == 2147483648, 1

又和答案一致了！

-2147483647-1 < 2147483647, 1

-2147483647-1U < 2147483647, 0

-2147483647-1 < -2147483647, 1

-2147483647-1U < -2147483647, 1

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.3, C语言中的类型转换

课本pg. 41, 2.2.2 节, 例题2.21

- 结论
 - gcc默认采用C99标准
- 经验教训
 - 凡事不如试一试
 - 永远考虑数据的机器级表示范围
 - 永远不要写出这种二义性的代码
 - 永远不要忽视warning

```
CFLAGS := -ggdb3 -MMD -MP -Wall -Werror -O2 -I./include -I../include
```

现在能理解为何在./nemu/Makefile的编译选项中，
加入-Wall和-Werror选项？

非数值型

char, char *

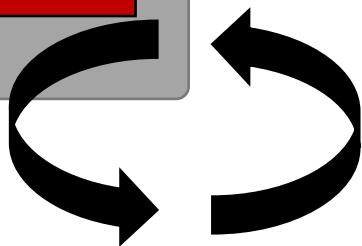
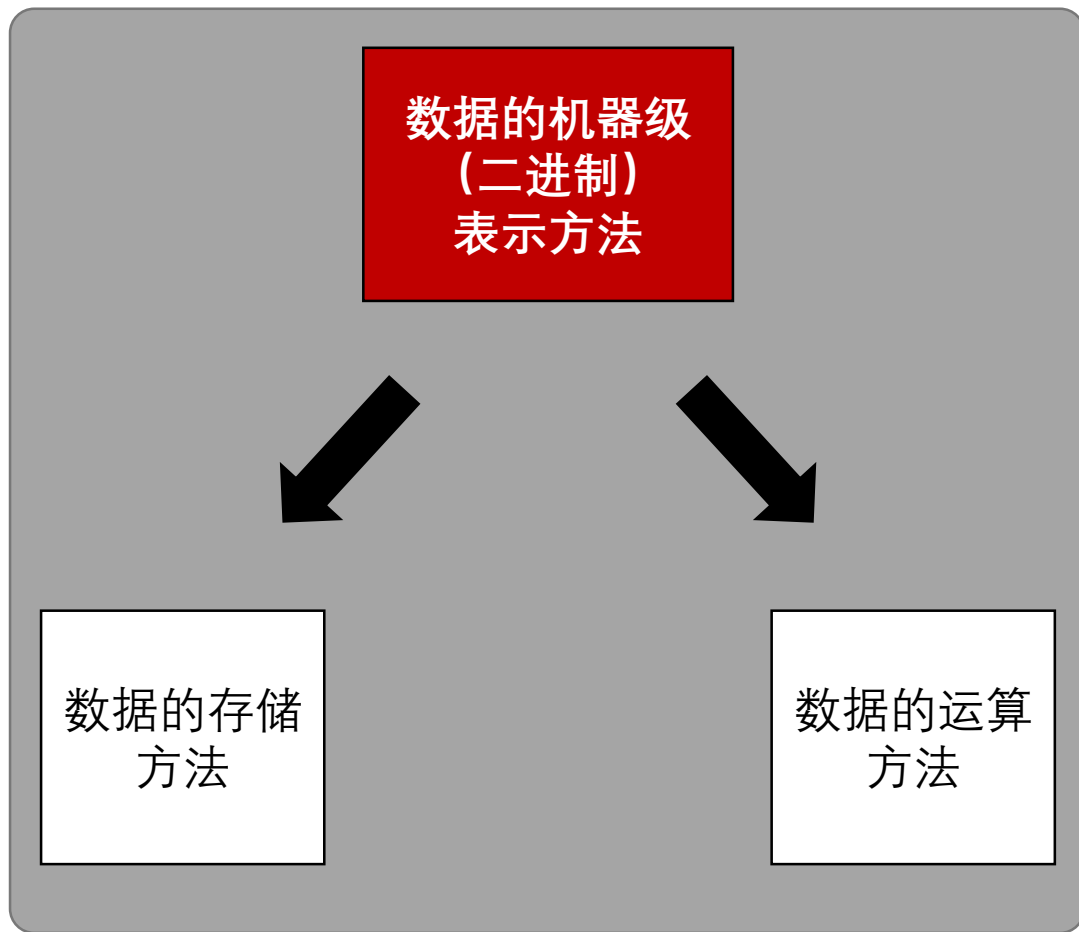
内容3：非数值型数据的表示

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

内容3：非数值型数据的表示

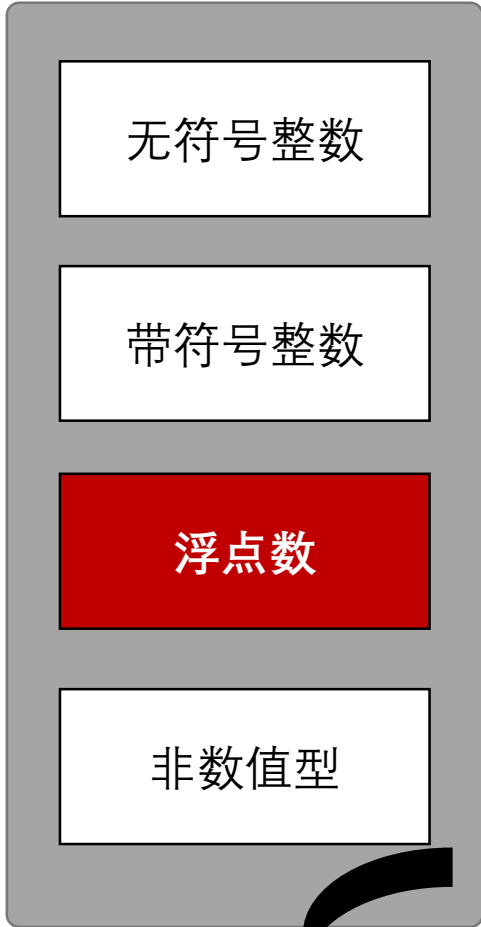
- 没啥好讲的
 - ASCII码表几类重要字符记住
 - 硬把char类型当作数值型也能计算
 - 其它的编码方式大概了解了解

浮点数

float, double

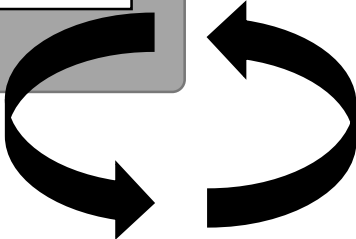
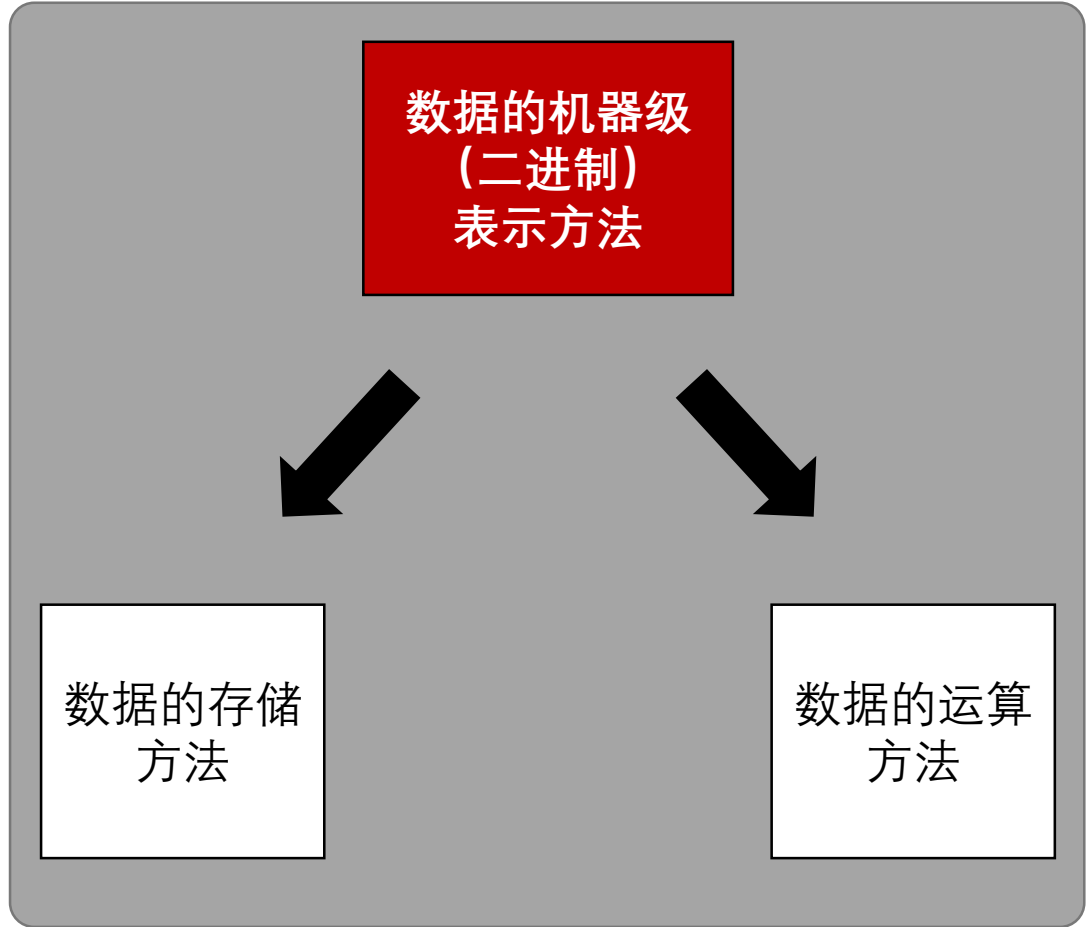
内容4.1：浮点数的IEEE 754表示法

1 数据的类型



X

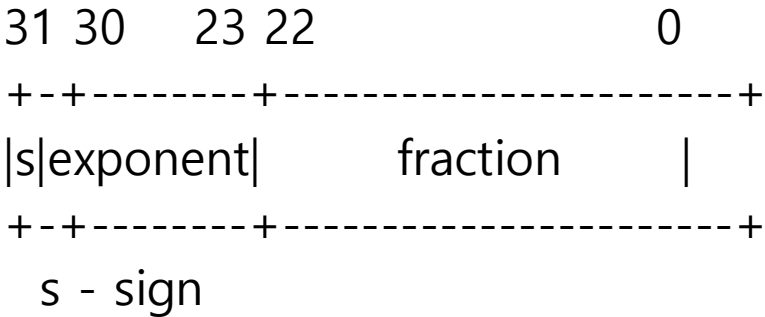
2 对应的问题



3 数据类型间的转换

内容4.1： 浮点数的IEEE 754表示法

Single-precision Floating Point



32位单精度浮点数的表示方法

0 表示正数 +
1 表示负数 -

假设是规格化数

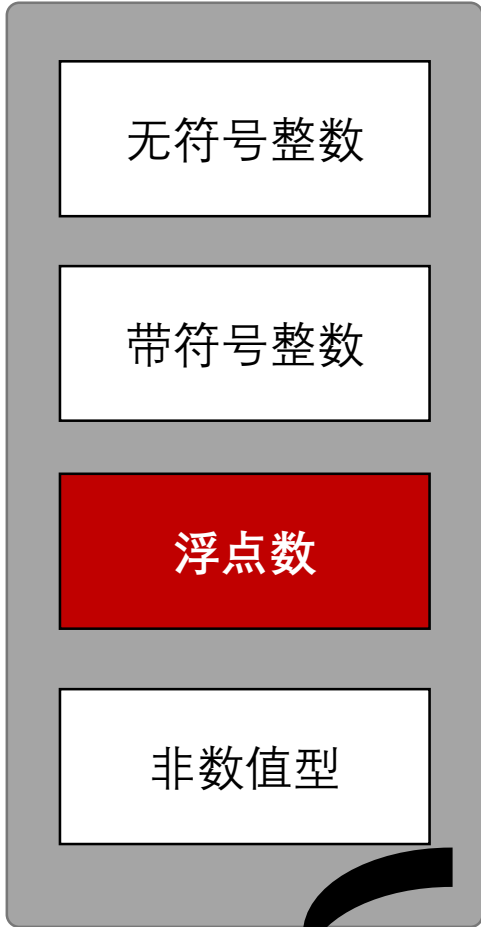
= s 1.fraction * 2^(exponent - 127)

阶码exponent - 偏置常数127
构成指数部分，注意非规格
化数exponent为0，但指数为
-126

尾数fraction部分加上隐藏位构成
significand（非规格化数隐藏位是0）

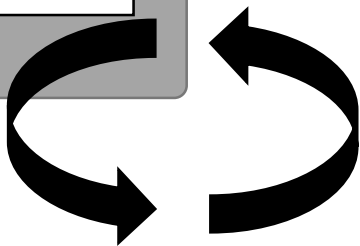
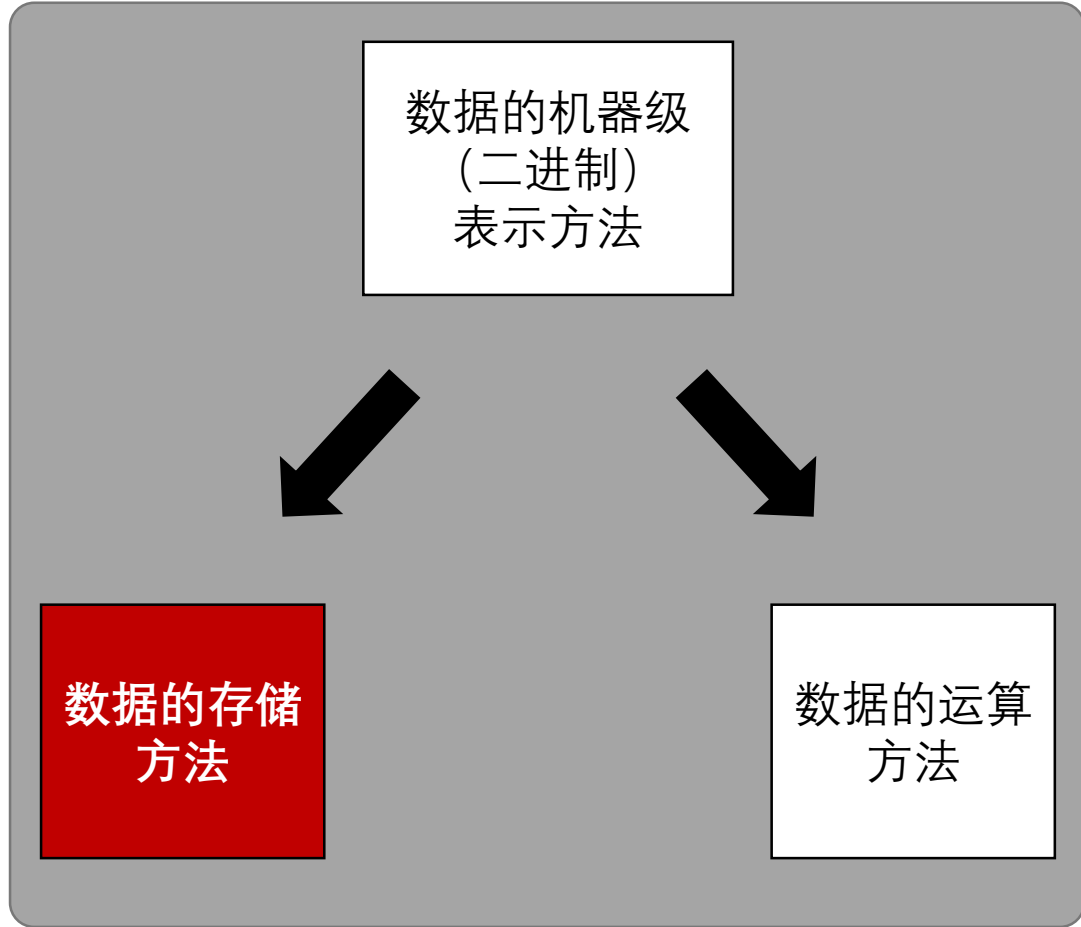
内容4.2：浮点数的存储

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

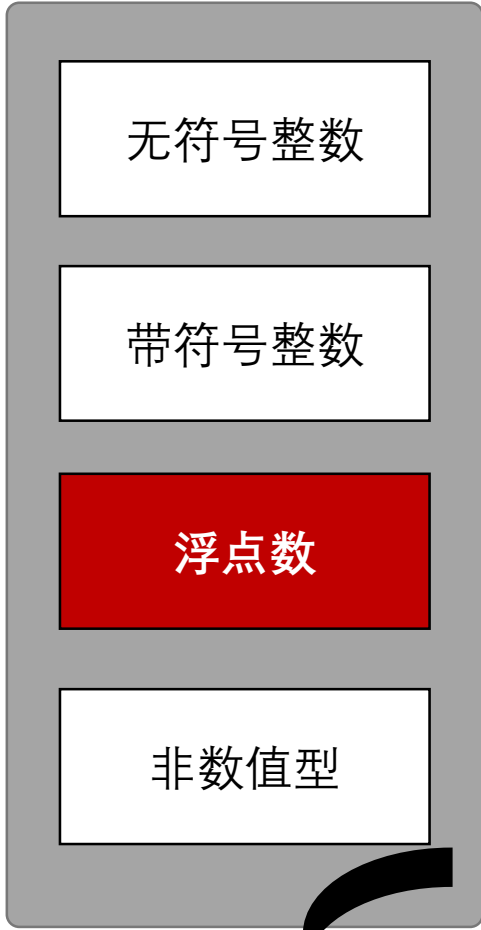
内容4.2：浮点数的存储

对应32位机器数，按照无符号整数的方法去存储

对应FPU内部有个浮点栈，有兴趣的同学自己去读读PA代码，搜搜手册资料（i386手册里没有）

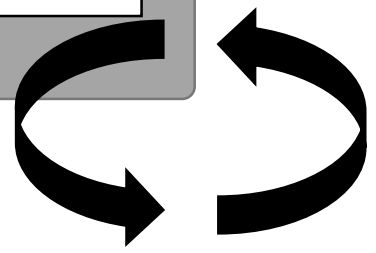
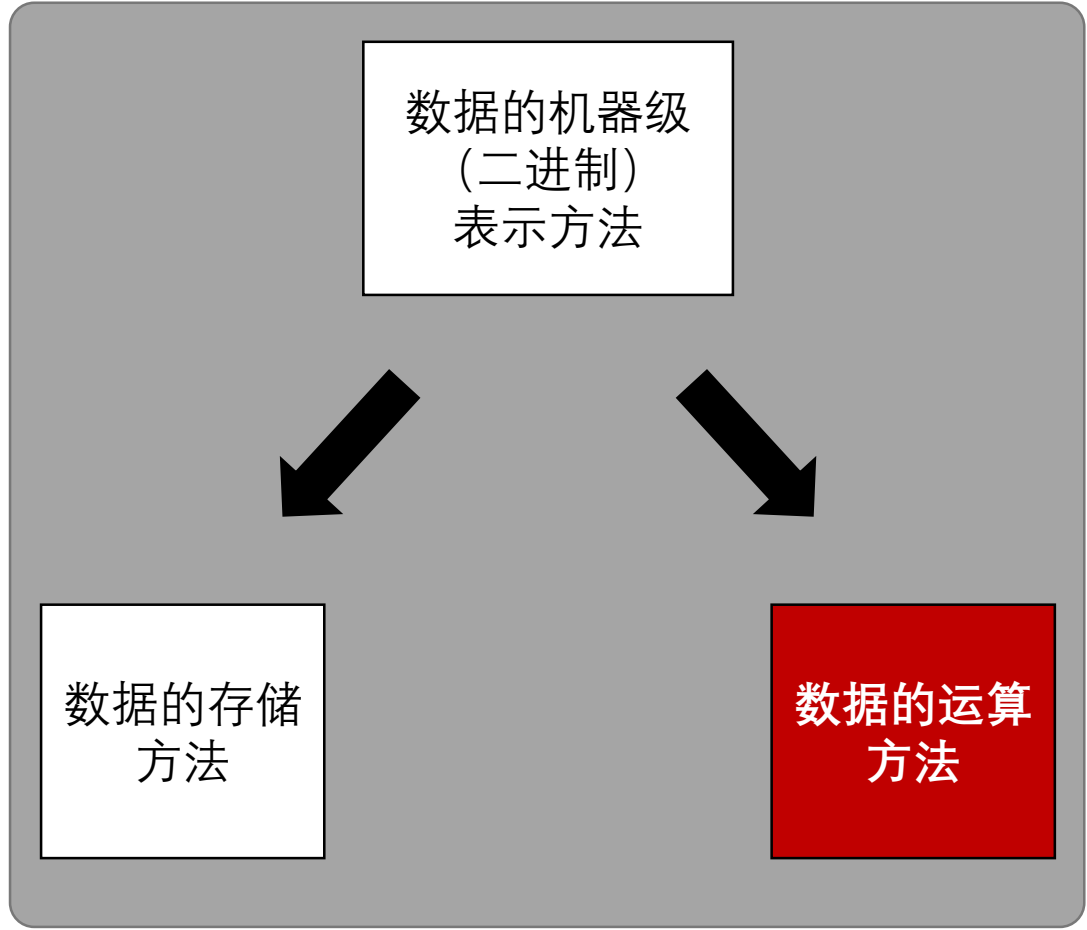
内容4.3：浮点数的运算（结合PA讲）

1 数据的类型



X

2 对应的问题



3 数据类型间的转换

下接汇编语言的简介