

数据表示

元培数科 王恩博

Let's get started →

2025/9/17



小班安排更新

周次	日期	大班课	展示同学	日期	对应大班节数	日期	大班课	展示同学
	周一	5~6节		周三	10~11节	周四	5~6节	
第1周	9月8日	1		9月10日	1	9月11日	2	龚鹤宁 陈泽羽
第2周	9月15日	3	孔令珊 刘筱	9月17日	2	9月18日	4	张哲涵 黎祖含
第3周	9月22日	5	陈俊宇 胡勋炀	9月24日	3/4	9月25日	6	黎宣萱 李準珩
第4周	9月29日	7	陈俊言 胡思成	10月1日	国庆放假	10月2日	国庆放假	
第5周	10月6日	国庆放假		10月8日	国庆放假	10月9日	8	王欣语 曾若天
第6周	10月13日	9	阶段测验1 (第1~8课)	10月15日	5/6/7	10月16日	10	龚鹤宁 陈泽羽
第7周	10月20日	11	孔令珊 刘筱	10月22日	8/10	10月23日	12	张哲涵 黎祖含
第8周	10月27日	13	陈俊宇 胡勋炀	10月29日	11/12	10月30日	14	黎宣萱 李準珩
第9周	11月3日	15	陈俊言 胡思成	11月5日	13/14	11月6日	16	王欣语 曾若天
第10周	11月10日	17	龚鹤宁 陈泽羽	11月12日	15/16	11月13日	18	孔令珊 刘筱
第11周	11月17日	19	阶段测验2 (第10~18课)	11月19日	17/18	11月20日	20	张哲涵 黎祖含
第12周	11月24日	21	陈俊宇 胡勋炀	11月26日	20	11月27日	22	黎宣萱 李準珩
第13周	12月1日	23	陈俊言 胡思成	12月3日	21/22	12月4日	24	王欣语 曾若天
第14周	12月8日	25	待定	12月10日	23/24	12月11日	26	待定
第15周	12月15日	27	LAB测验 (L1~L7)	12月17日	25/26	12月18日	28	待定
第16周	12月22日	29		12月24日	28	12月25日	30	
考试周	12月29日		期末考试 (周一下午)					

基本概念

basic concepts

位 (bit) : 数据存储的最小单位

字节 (Byte) : 最小的可寻址的存储器单位, 1

Byte = 8 bits

十六进制

- 0x 开头
- 0~9, A~F , 不区分大小写
- 一个字节的值域是 $00_{16} \sim FF_{16}$

$0xA = 10 = 1010$

$0xC = 12 = 1100$

$0xE = 14 = 1110$

*需要熟练掌握进制转换

基本概念

basic concepts

字长 (word size) : 决定虚拟地址空间的最大大小

- 对于一个字长为 w 位的机器而言, 虚拟地址的范围为 $0 \sim 2^{w-1}$, 程序最多访问 2^w 个字节
- 为什么? 我们需要计算机去寻址, 从而需要用 w 位二进制数去表示地址, 所以地址的个数就是 2^w , 所以程序最多访问 2^w 个字节

字节顺序

- 小端序 (little endian) : 最低有效字节在最前面
- 大端序 (big endian) : 最高有效字节在最前面

一定要先想清楚字节是怎么排序的 (哪边是小地址哪边是高地址), 再去辨析大端/小端序

字节是一个整体, 不会说字节内部是按照小端序还是大端序排列

字节顺序

byte order

同样是表示 `0x12345678`，在不同的机器上，内存中的存储可能是：

- 小端序： (低地址) `0x78 0x56 0x34 0x12` (高地址)
- 大端序： (低地址) `0x12 0x34 0x56 0x78` (高地址)

看到了吗，字节内部顺序是固定的，和大端序小端序无关！

一定要在脑子里明确知道，所谓内存，就是一系列连续的字节。无论你是现在初学的一维线性模型，还是后面做 lab 时会考虑到的二维模型，首先都要明确那边是低地址那边是高地址。

应用

- 现今，大多数计算机都采用 **小端序**。
- 网络通讯中，一般采用 **大端序**。
- 自动转换：`ntohs()` `ntohl()` `htons()` `htonl()`

字节顺序

byte order

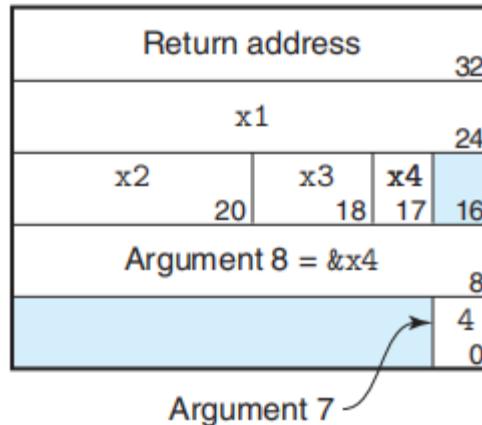
在这张图中，显示了一个内存的排布顺序。

- 同一行中，地址连续，越靠近右侧地址越小
- 同一列中，地址不连续，越靠近下方地址越小
- 也就是说，呈现出了一个 Z 字形

未来，你们可能会见识到各种内存排布方式，但无论如何，都要先搞清楚内存的排布方式，再去辨析大端序小端序。

全部内存 - 一块连续的内存空间（指令/数据） - 大端序/小端序 - 字节

当讨论的这一块内存空间只有 1 个字节（例如：一个 `char` 类型）时，我们还需要考虑大端序小端序吗？
NO!



布尔运算

boolean operation

`~ (not) 非`

0	1
---	---

1	0
---	---

`| (or) 或`

0	1
---	---

0	0	1
---	---	---

1	1	1
---	---	---

`& (and) 与`

0	0	0
---	---	---

1	0	1
---	---	---

`^ (eor) 异或`

0	1
---	---

0	0	1
---	---	---

1	1	0
---	---	---

逻辑运算

logical operation

- 逻辑运算符: `&&` (and, 与)、`||` (or, 或)、`!` (not, 非)
- 逻辑运算符的运算结果只有两种: `true` 和 `false`

注意与位运算区分! 逻辑运算具有短路特性, 位运算没有!

短路特性

- `&&` : 前一个表达式为假时, 后一个表达式不执行, 因为无论后一个表达式是什么, 结果都是假
- `||` : 前一个表达式为真时, 后一个表达式不执行, 因为无论后一个表达式是什么, 结果都是真

也即: 如果对第一个参数求值能确定表达式的结果, 那么逻辑运算符不会对第二个参数求值

Q: 为什么 `p && *p++` 不会引用空指针? (注意这里也有短路特性)

A: 因为 `&&` 运算符的短路特性, 当 `p` 为空指针时 (对应其数字表达为 `0x00000000`, 或者说是 `NULL`) , `*p` 不会被执行, 所以不会出现引用空指针的情况。

位移

bit shift

- 左移: <<
- 右移: >>

逻辑右移 / 算术右移

- 逻辑右移: 左边补 0
- 算术右移: 左边补 最高位 (思考: 为什么?)
- 逻辑左移 / 算术左移: 右边补 0

C 语言: 有符号数算术右移, 无符号数逻辑右移

$a << 3 + b << 2$ 运算时, 需要注意运算符优先级
(如果记不住就全加上括号!)

优先级	运算符	结合律	助记
1	::	从左至右	作用域
2	a++、a--、 type()、type{}、 a()、a[]、 .、->	从左至右	后缀自增减、 函数风格转型、 函数调用、下标、 成员访问
3	!、~、 ++a、--a、+a、-a、 (type)、sizeof、&a、 *a、 new、new[]、delete、delete[]	从右至左	逻辑非、按位非、 前缀自增减、正负、 C 风格转型、取大小、取址、 指针访问、 动态内存分配
4	.*、->*	从左至右	指向成员指针
5	a*b、a/b、a%b	从左至右	乘除、取模
6	a+b、a-b	从左至右	加减
7	<<、>>	从左至右	按位左右移
8	<、<=、>、>=	从左至右	大小比较
9	==、!=	从左至右	等价比较
10	a&b	从左至右	按位与
11	^	从左至右	按位异或
12	,	从左至右	从左至右
13	&&	从左至右	逻辑与
14	,	从左至右	,
15	a?b:c、 =、+=、-=、*=、/=、%=、&=、^=、 =、<=>=	从右至左	=、<=>=
16	,	从左至右	逗号

整数表示

Symbol	Type	Meaning	Page
$B2T_w$	Function	Binary to two's complement	100
$B2U_w$	Function	Binary to unsigned	98
$U2B_w$	Function	Unsigned to binary	100
$U2T_w$	Function	Unsigned to two's complement	107
$T2B_w$	Function	Two's complement to binary	101
$T2U_w$	Function	Two's complement to unsigned	107
$TMin_w$	Constant	Minimum two's-complement value	101
$TMax_w$	Constant	Maximum two's-complement value	101
$UMax_w$	Constant	Maximum unsigned value	99
$+^t_w$	Operation	Two's-complement addition	126
$+^u_w$	Operation	Unsigned addition	121
$*^t_w$	Operation	Two's-complement multiplication	133
$*^u_w$	Operation	Unsigned multiplication	132
$-^t_w$	Operation	Two's-complement negation	131
$-^u_w$	Operation	Unsigned negation	125

整数编码

integer encoding

无符号数

对向量 $x = [x_{w-1}, x_{w-2}, \dots, x_0]$:

$$\text{B2U}_w(x) = \sum_{i=0}^{w-1} (x_i \times 2^i)$$

- B2U_w 表示把二进制编码转化为非负整数
(Binary to Unsigned)

该函数是双射，即无符号数编码具有唯一性

- $\text{UMax}_w = 2^w - 1$

有符号数

对向量 $x = [x_{w-1}, x_{w-2}, \dots, x_0]$:

$$\text{B2T}_w(x) = -x_{w-1} \times 2^{w-1} + \sum_{i=0}^{w-2} (x_i \times 2^i)$$

- B2T_w 表示把二进制编码转化为补码
(Binary to Two's Complement)
- 当最高位为 1，整个数表示为一个负数
- 同样具有唯一性

整数编码

integer encoding

对于有符号数，取值范围是不对称的，典型例子 int：

- $TMax_{32} = 2147483647$
- $TMin_{32} = -2147483648$

为什么？

数据的其他表示

other representations

- **补码**: 上下文无关——同构环

$$B2T_w(\vec{x}) = -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

- **反码**: 0 的表示有两种: 000...0 和 111...1

$$B2O_w(\vec{x}) = -x_{w-1}(2^{w-1} - 1) + \sum_{i=0}^{w-2} x_i 2^i$$

- **原码**: 最高位是符号位用来确定剩下的是正数还是负数 (e.g. 浮点数)

$$B2S_w(\vec{x}) = (-1)^{x_{w-1}} \cdot \sum_{i=0}^{w-2} x_i 2^i$$

有符号数与无符号数之间的转换

conversion between signed and unsigned

强制类型转换保持位值不变，只改变解释这些位的方式

补码→无符号数

对满足 $TMin_w \leq x \leq TMax_w$ 的 x 来说：

- 若 $TMin_w \leq x < 0$, 则 $T2U(x) = x + 2^w$
- 若 $0 \leq x \leq TMax_w$, 则 $T2U(x) = x$

无符号数→补码

对满足 $0 \leq x \leq UMax_w$ 的 x 来说：

- 若 $x > TMax_w$, 则 $U2T(x) = x - 2^w$
- 若 $0 \leq x \leq TMax_w$ 则 $U2T(x) = x$

对于负数，最高位“翻转了阵营”，从补码的代表的 -2^{w-1} 变成了 $+2^{w-1}$

当一个无符号数与一个有符号数进行计算时，有符号数在这个表达式中会被当做无符号数（即发生了隐式的强制类型转换）

例如： $-1 > 0u$ 此为无符号的比较

(-1 在计算机中表示为全 1, 即 $UMax_w$, 而 $0u$ 表示为全 0 的无符号数)

有符号数与无符号数之间的转换

conversion between signed and unsigned

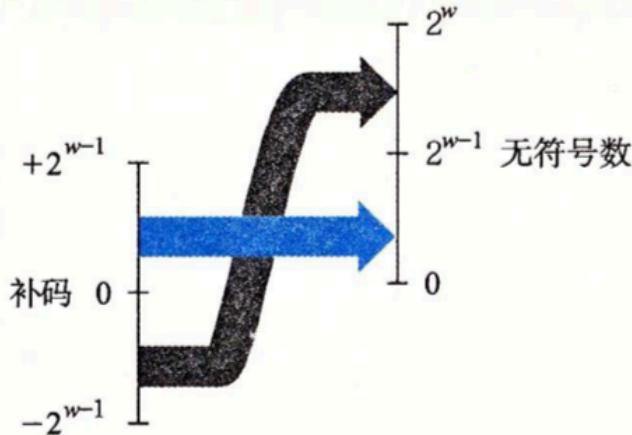


图 2-17 从补码到无符号数的转换。函数 $T2U$ 将负数转换为大的正数

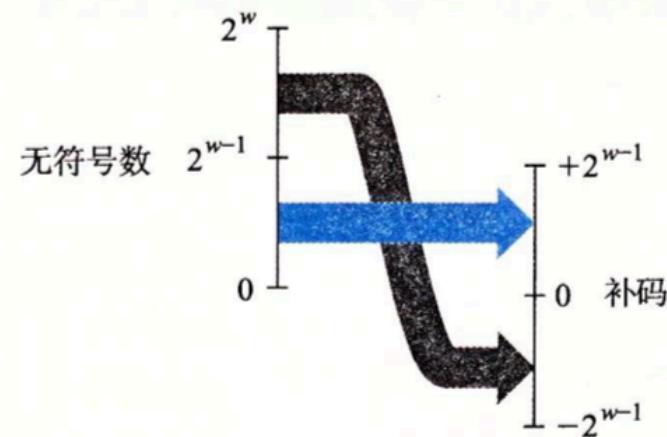


图 2-18 从无符号数到补码的转换。函数 $U2T$ 把大于 $2^{w-1}-1$ 的数字转换为负值

扩展和截断

extension and truncation

扩展一个数

- 对于无符号数，高位补 0
- 对于有符号数（补码），高位补符号位

当把 short（有符号数，2 字节）强制转换为 unsigned（无符号数，4 字节）时？

先进行数位扩展，再转换为无符号数

为什么对于有符号数，高位补符号位？

Hint: 消消乐！

设现在位数为 m ，补到 n 位，且 $n > m$ ，则：

- 若最高位为 0，显然；
- 若最高位为 1，则你补的一堆 1 和原先的 1 等价

$$-2^{n-1} + \sum_{i=m-1}^{n-2} 2^i = -2^{m-1}$$

扩展和截断

extension and truncation

截断一个数

- 对于无符号数 $x = [x_{w-1}, \dots, x_0]$, 截断为 w' 位, 则 $x' = [x_{w'-1}, \dots, x_0]$
即 $x' = x \bmod 2^k$
- 补码截断, 原理上与无符号数类似, 但对于数位的解释方式不同

整数加减法

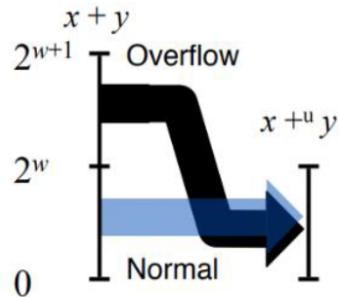
integer addition and subtraction

无符号数加法

由于两个 w 位的无符号数相加，结果的范围是 $[0, 2^{w+1} - 2]$ ，而这需要 $w + 1$ 位来表示，所以实际上：

结果表示 $x + y$ 的低 w 位，也即 $x + y \bmod 2^w$

- 当 $x + y < 2^w$ 时，结果正确；
- 当 $x + y \geq 2^w$ 时， $x + y$ 的结果需要减去 2^w ，才会得到实际结果，即 $x + y - 2^w$ ，这就是 **溢出**



无符号数加法

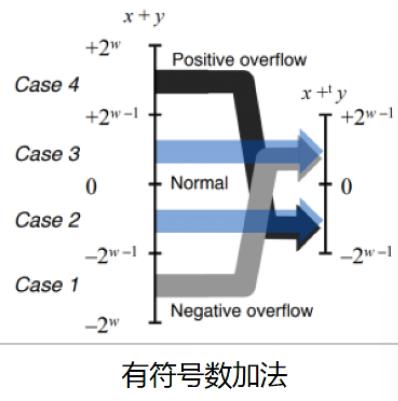
整数加减法

integer addition and subtraction

有符号数加法

由于两个 w 位的有符号数相加，结果的范围是 $[-2^w, 2^w - 2]$ ，所以实际上：

- 若 $-2^{w-1} \leq x + y < 2^{w-1}$ ，结果正确；
- 若 $x + y \geq 2^{w-1}$ ，结果需要减去 2^w ，才会得到实际结果，即 $x + y - 2^w$ ，此时称为 **正溢出/上溢出**
- 若 $x + y < -2^{w-1}$ ，结果需要加上 2^w ，才会得到实际结果，即 $x + y + 2^w$ ，此时称为 **负溢出/下溢出**



判断溢出

determine overflow

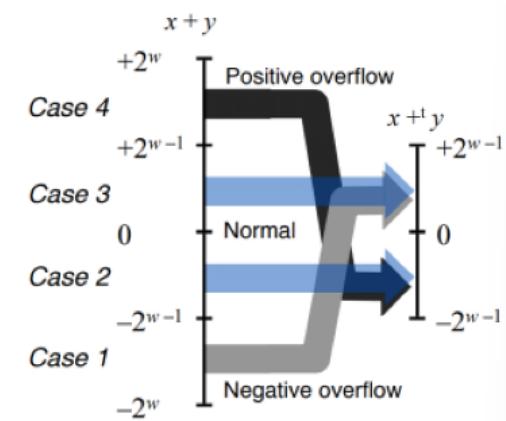
无符号数加法

- $s = x + y$, 若 $s < x$ 或 $s < y$ 则溢出

有符号数加法

- $s = x + y$, 若 $x > 0$ 且 $y > 0$ 且 $s \leq 0$, 则正溢出;
- 若 $x < 0$ 且 $y < 0$ 且 $s \geq 0$, 则负溢出

为什么只有这两种情况?



有符号数加法

判断溢出

determine overflow

关于整数加减法/溢出的一个有趣事实：这里的加法是个 **环**（阿贝尔群）！

无论你怎么改变次序，它总是可以轮换回来~

这是一个做题很方便的技巧！

- **补码的非**（加法逆元）：

- 数学公式

$$-_w^t x = \begin{cases} TMin_w & x = TMin_w \\ -x & x > TMin_w \end{cases}$$

- 计算机实现(位级表示)

$$\begin{aligned} -x &= \sim x + 1 \\ \text{because : } x + (-x) &=_w^u 2^w \end{aligned}$$

证明: $\sim x + 1 = -x \quad (x \neq TMin)$

证:

$$\sim x + 1 = -x$$

$$\sim x + x = -1$$

证明: $TMax + 1 = TMin$

证: 想想消消乐

$$0111\dots1111 + 1 = 1000\dots0000$$

对于 $TMin$

$$\sim TMin = 0111\dots1111$$

$$\sim TMin + 1 = 1000\dots0000$$

从数学上: $-TMin = 01000\dots000$ 截断之后仍为
 $TMin$

补码的非运算

two's complement negation

对满足 $TMin_w \leq x \leq TMax_w$ 的 x 来说：

- 若 $x = TMin_w$, 则 $-x = TMin_w$
- 否则, $-x = -x$ (算术上的)

注：这里式子左边的 $-x$ 是 **算数逆元**

- 对于 $x \neq TMin_w$ 时, 本身 $-x$ 也在表示范围内, 结论是平凡的
- 对于 $x = TMin_w$ 时, $-x$ 不在表示范围内。

但是, 两个 $TMin_w$ 相加, 溢出到了上一位, 结果是 0, 所以此时我们说 $TMin_w$ 的算数逆元就是它自身

回忆：刚才说的 $\sim x + 1 = -x$

乘除法

multiplication and division

无符号数乘法

等于乘法计算得到的结果（一个 $2w$ 位长的数），而后截断到 w 位

补码乘法

等于有符号数乘法后得到的结果（一个 $2w$ 位长的数），无符号截断到 w 位，而后将最高位转化为符号位

核心：粗暴的位级表示截断

乘以二的幂次

此时，相当于左移

乘除法

multiplication and division

特别注意：除法这里以二的幂次作为除数，也只有此时可以采用右移来取巧。

无符号数除法

无符号数 u 除以 2^k ：将 u 逻辑右移 k 位

(此时恰好就是舍入到 0 的)

有符号数除法

直接右移 $x \gg k$ 得到的是 向下舍入 的结果，而不是正常来讲的向 0 取整。

例如： $-3 / 2 = -1$ ，而 $-3 \gg 1 = -2$

向 0 取整： $(x < 0) ? (x + (1 << k) - 1 : x) \gg k$

证明见书 P73

作业讲评

P88 2.59, 2.60

** 2.59 编写一个 C 表达式，它生成一个字，由 x 的最低有效字节和 y 中剩下的字节组成。对于运算数 $x = 0x89ABCDEF$ 和 $y=0x76543210$ ，就得到 $0x765432EF$ 。

** 2.60 假设我们将一个 w 位的字中的字节从 0(最低位)到 $w/8-1$ (最高位)编号。写出下面 C 函数的代码，它会返回一个无符号值，其中参数 x 的字节 i 被替换成字节 b :

```
unsigned replace_byte (unsigned x, int i, unsigned char b);
```

以下示例，说明了这个函数该如何工作：

```
replace_byte(0x12345678, 2, 0xAB) --> 0x12AB5678  
replace_byte(0x12345678, 0, 0xAB) --> 0x123456AB
```

根据题目给出的示例，这里认为使用32位机器，一个字大小为4字节。

```
unsigned concat_byte(unsigned x, unsigned y){  
    return (x & 0xFF) | (y & 0xFFFFF00);  
}
```

作业讲评

P88 2.59, 2.60

** 2.59 编写一个 C 表达式，它生成一个字，由 x 的最低有效字节和 y 中剩下的字节组成。对于运算数 $x = 0x89ABCDEF$ 和 $y=0x76543210$ ，就得到 $0x765432EF$ 。

** 2.60 假设我们将一个 w 位的字中的字节从 0(最低位)到 $w/8-1$ (最高位)编号。写出下面 C 函数的代码，它会返回一个无符号值，其中参数 x 的字节 i 被替换成字节 b :

```
unsigned replace_byte (unsigned x, int i, unsigned char b);
```

以下示例，说明了这个函数该如何工作：

```
replace_byte(0x12345678, 2, 0xAB) --> 0x12AB5678  
replace_byte(0x12345678, 0, 0xAB) --> 0x123456AB
```

```
unsigned replace_byte(unsigned x, int i, unsigned char b)  
{  
    unsigned mask = 0xFF << (i * 8);  
    unsigned shifted_byte = (unsigned)b << (i * 8);  
    return (x & ~mask) | shifted_byte;  
}
```

作业讲评

P91 2.71

- * 2.71 你刚刚开始在一家公司工作，他们要实现一组过程来操作一个数据结构，要将 4 个有符号字节封装成一个 32 位 unsigned。一个字节中的字节从 0(最低有效字节)编号到 3(最高有效字节)。分配给你的任务是：为一个使用补码运算和算术右移的机器编写一个具有如下原型的函数：

```
/* Declaration of data type where 4 bytes are packed
   into an unsigned */
typedef unsigned packed_t;

/* Extract byte from word.  Return as signed integer */
int xbyte(packed_t word, int bytenum);
```

也就是说，函数会抽取出指定的字节，再把它符号扩展为一个 32 位 int。

你的前任(因为水平不够高而被解雇了)编写了下面的代码：

```
/* Failed attempt at xbyte */
int xbyte(packed_t word, int bytenum)
{
    return (word >> (bytenum << 3)) & 0xFF;
```

- A. 这段代码错在哪里？
- B. 给出函数的正确实现，只能使用左右移位和一个减法。

作业讲评

P91 2.71

A : 这段代码错在，当获取的 `unsigned int` 字节的最高位上为1的时候，转换后的 `int` 应为负数，但这一算法得到的结果却是正数。

B :

```
int xbyte(packed_t word, int bytenum) {
    // 先将word左移，使word最高位上是所要提取byte的最高位
    int shifted = (int)(word << ((3 - bytenum) << 3));
    // 再执行右移运算，高位自动填入1或0，实现正负
    return shifted >> 24;
}
```

TIPS

datalab...

认真看readme

实在想不出也没关系，这些tricks用处不大

一个数学tips

- $(a + b) \bmod c = [(a \bmod c) + (b \bmod c)] \bmod c$
- 设 c 与 m 互素，则 $a \equiv b \pmod{m}$ 当且仅当 $ca \equiv cb \pmod{m}$

习题试炼 1

2. 在采用小端法存储机器上运行下面的代码，输出的结果将会是？

(int,unsigned 为 32 位长,short 为 16 位长,0~9 的 ASCII 码分别是 0x30~0x39)

```
char *s = "2018";
int *p1 = (int *)s;
short s1 = (*p1) >> 12;
unsigned u1 = (unsigned) s1;
printf("0x%x\n", u1);
```

- A) 0x00002303
- B) 0x00032303
- C) 0xffff8313
- D) 0x00008313

习题试炼 1

答案：C

本题考查大小端存储，以及整数类型转化。字符串在存储时不区分大小端，前面的字符存储在低地址，而在被转化成整型的时候低地址被视为低位，因此*p1 为 0x38313032，s1 为 0x8313。在由 short 转化为 unsigned 的时候，我们要先改变大小，之后完成有符号到无符号的转换，因此 u1 为 0xffff8313。

习题试炼 2

3. 运行下面的代码，输出结果是（其中 float 类型表示 IEEE-754 规定的浮点数，包括 1 位符号、8 位阶码和 23 位尾数）：

```
for(float f = 1;; f = f + 1)
{
    if(f + 1 - f != (float)1)
    {
        printf("%.0f\n", f);
        break;
    }
}
```

- A. 8388608 ($=2^{23}$)
- B. 16777216 ($=2^{24}$)
- C. 2147483647 ($=2^{31}-1$)
- D. 程序为死循环，没有输出

习题试炼 2

答案:B

解答: 这种 float 类型的 23 位尾数意味着其与 23 位及以内的整数可以建立一一对应的联系, 自然 $f + 1 - f == 1$ 成立; 当 f 超过这一范围时, $f + 1$ 的运算结果会发生舍入, 因而无法恢复到之前的结果。随着 f 自增, 最小的超过精度范围的整数是 2^{24} , 故可以得出选项。

习题试炼 3

4. 在 x86-64 机器上, 有如下的定义:

```
int x = _____;
int y = _____;
unsigned int ux = x;
unsigned int uy = y;
```

判断下列表达式是否等价:

(提示: 减法的运算优先级比按位异或高。布尔运算的结果都是有符号数。)

	表达式 A	表达式 B	等价吗?
(1)	$x > y$	$ux > uy$	Y N
(2)	$(x > 0) \ \ (x < ux)$	1	Y N
(3)	$x \wedge y \wedge x \wedge y \wedge x$	x	Y N
(4)	$((x >> 1) << 1) \leq x$	1	Y N
(5)	$((x / 2) * 2) \leq x$	1	Y N
(6)	$x \wedge y \wedge (\sim x) - y$	$y \wedge x \wedge (\sim y) - x$	Y N
(7)	$(x == 1) \ \&\ (ux - 2 < 2)$	$(x==1) \ \&\ ((\sim ux) - 2 < 2)$	Y N

习题试炼 3

提示：减法的运算优先级比按位异或高。布尔运算的结果都是有符号数。

	表达式 A	表达式 B	等价吗？
(1)	$x > y$	$ux > uy$	Y N
(2)	$(x > 0) \mid\mid (x < ux)$	1	Y N
(3)	$x \wedge y \wedge x \wedge y \wedge x$	x	Y N
(4)	$((x >> 1) << 1) \leq x$	1	Y N
(5)	$((x / 2) * 2) \leq x$	1	Y N
(6)	$x \wedge y \wedge (\sim x) - y$	$y \wedge x \wedge (\sim y) - x$	Y N
(7)	$(x == 1) \&\& (ux - 2 < 2)$	$(x==1) \&\& ((\sim ux) - 2 < 2)$	Y N

【答】(1) 取 $x=1, y=-1$ 即不正确；(2) 取 $x=-1$ 即不正确；(3) 正确，利用交换律、结合律，以及 $x \wedge x == 0$ ；(4) 正确，即使是对负数；(5) 不正确，负奇数该运算向 0 舍入；(6) 正确， $(\sim x) - y$ 也就是 $(\sim x) + (\sim y) + 1$ ，注意运算优先级；(7) 不正确， $\sim ux$ 是有符号数。

习题试炼 4

18. 若我们采用基于 IEEE 浮点格式的浮点数表示方法, 阶码字段 exp 占据 k 位, 小数字段 frac 占据 n 位, 则最大的非规格的正数是 _____ (结果用含有 n, k 的表达式表示)

习题试炼 4

答案: $(1 - 2^{-n}) * 2^{-2^{k-1} + 2}$

解析: 阶码字段 exp: 00...00

小数字段 frac: 11...11

偏置值 bias: $2^{k-1} - 1$

习题试炼 5

4. 关于浮点数，以下说法正确的是
- A. 给定任意浮点数 a, b 和 x ，如果 $a > b$ 成立（求值为 1），则一定 $a+x > b+x$ 成立
 - B. 不考虑结果为 NaN 、 Inf 或运算过程发生溢出的情况，高精度浮点数一定得到比低精度浮点数更精确或相同的结果
 - C. 不考虑输入为 NaN 、 Inf 的情况，高精度浮点数一定得到比低精度浮点数更精确或相同的结果
 - D. 给定任意浮点数 a, b 和 x ，如果 $a > b$ 不成立（求值为 0），则一定 $a+x > b+x$ 不成立。

习题试炼 5

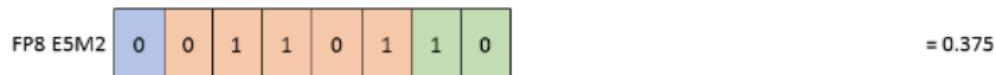
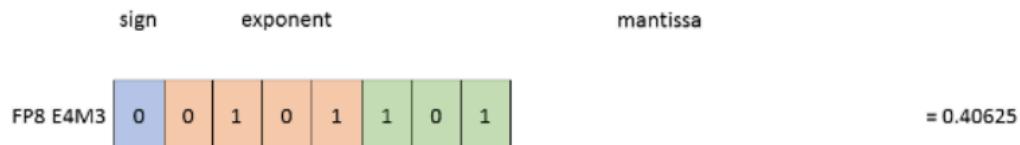
答案: d。如果不出现 NaN 和 Inf, 浮点数是满足单调性的, 这时 a 和 d 都成立。如果出现 NaN 和 Inf, 那么任何时候比较运算的操作数有 NaN 和 Inf 就为 0, 就只有 d 成立。关于 b 和 c, 由于运算的偶然性, 高精度不一定比低精度精确。

习题试炼 6

第二题（15 分） 请结合教材第二章的相关知识，回答下列问题

量化 (Quantization) 是将输入值从大集合映射到较小集合中的输出值的过程，在神经网络中是压缩模型的经典方法，通过降低权重、偏差和激活精度消耗更少的内存。量化利用了神经网络对噪声的弹性；通过使用较低精度的值，量化会引入一些噪声，但深度神经网络仍然可以学习拾取关键模式并忽略噪声。在 Nvidia H100 GPU 中引入了对新数据类型 FP8 (8 位浮点) 的支持，从而实现了更高的矩阵乘法和卷积吞吐量。FP8 包含 E4M3 和 E5M2 两种不同的数据类型，E4M3 包含 1 位符号，4 位阶码，3 位尾数；E5M2 包含 1 位符号，5 位阶码，2 位尾数。

（M 是 mantissa 的缩写表示尾数部分，与书本上 fraction 等价）



习题试炼 6

	sign	exponent				mantissa			
FP8 E4M3	0	0	1	0	1	1	0	1	= 0.40625
FP8 E5M2	0	0	1	1	0	1	1	0	= 0.375

1. 其中 E5M2 遵守 IEEE-754 标准，根据所学的 float 和 double 的知识，该数据类型能表示最大的规格化数（二进制）是(1) ____；最大的非规格化数（二进制）是(2) ____。

2. 但是在 Nvidia 的设计中 E4M3 放弃了 IEEE-754 标准，通过回收用于特殊值的大多数位模式来扩展动态范围，它只保留了阶码 (E) 和尾数 (M) 全为 1 时为特殊值 NaN，则该数据类型将能表示的最大规格化数扩展到（二进制）(3) _____，值（十进制）是(4) _____，是 IEEE-754 标准设计出的最大规格化数的(5) _____ 倍（舍入到整数）。因为在这种情况下，实现的更大范围比支持特殊值的多种编码有用得多。

习题试炼 6

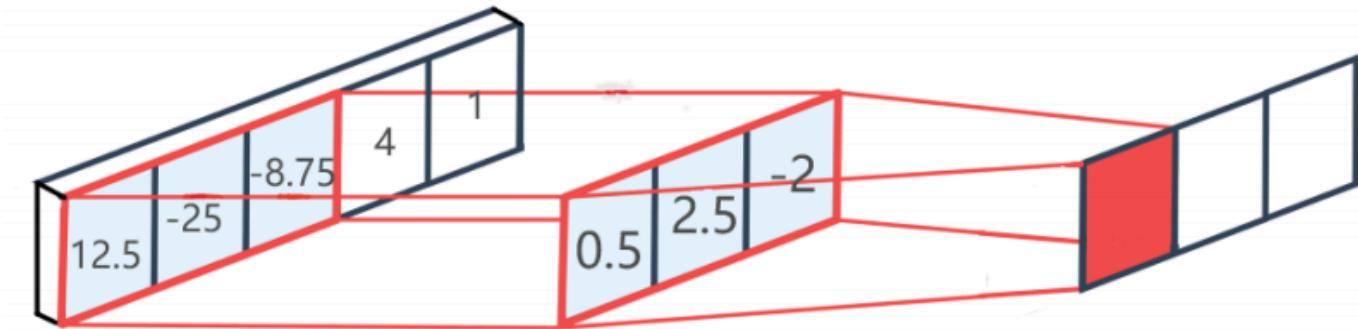
3. 另一种 8bit 格式 int8 (8bit 整型) 在模型量化中占有重要地位。与 float 相比, int8 在少量精度损失下将模型大小和内存带宽需求降低 4 倍, 使得模型表示更加紧凑, 并且在 PyTorch, TensorFlow 框架中都有很好的支持。那么在 H100 GPU 上用 Nvidia E4M3 表示 int8, 是否会发生溢出 (6) _____

(请填是/否), 是否会发生舍入 (7) _____ (请填是/否)。

4. 深度学习模型计算会涉及大量浮点数的加法和乘法, 而计算顺序对于结果有很大的影响。要将 E5M2 表示的 -1.25 , 1.5 , $1/16$, $1/32$ 四个浮点数相加, 写出可能得到的最小计算结果 (分数) (8) _____, 最大计算结果 (分数) (9) _____。

习题试炼 6

5. 在深度学习中，我们期望使用在容忍一定误差的前提下使用内存占用最小的数据类型，则对于以下一维卷积应当采用 (10) _____ 的浮点型（请填 float,double/E4M3/E5M2）



在训练神经网络期间，通常前向激活和权重需要更高的精度，因此最好在前向传递期间使用 E4M3 数据类型。然而在后向传递中，流经网络的梯度通常不太容易受到精度损失的影响，但需要更高的动态范围。因此，最好使用 E5M2 数据格式存储它们。

习题试炼 6

答案：

1. $01111011_2; 00000011_2$ (2 分 2 分)
2. $01111110_2, 448, 2$ (2 分 1 分 1 分)
3. 不会, 会 (2 分 2 分)
4. $1/4, 3/8$ (向偶数舍入) (1 分 1 分)
5. E4M3 (1 分 1 分)

本题在深度学习模型量化压缩的场景下，主要考察对于 IEEE-754 标准的掌握和对浮点型指数部分(E)和底数部分(M)对于范围和精度的影响，并拓展考察在实际应用中浮点数计算顺序和选取适当浮点型的软件程序员需要注意和警惕的问题，引导同学们深入思考并将 ICS 的知识真正的学以致用

评分标准修订：

- (1) 写 1110000000000000 (13 个 0) 可以给 1 分，原来不得分

习题试炼 7

4. 阅读如下一段代码

```
int x = 0x2021;  
int y = -x;  
int z = (x / 2) - (x >> 1) + (y / 2) - (y >> 1);  
printf("%d", z);
```

问运行这段代码后的输出是什么：

习题试炼 7

4. 阅读如下一段代码

```
int x = 0x2021;  
int y = -x;  
int z = (x / 2) - (x >> 1) + (y / 2) - (y >> 1);  
printf("%d", z);
```

问运行这段代码后的输出是什么：

答案：1

习题试炼 8

2. 在 x86-64 机器上运行如下代码，输出是：

```
char A[12] = "19260817";
char B[12] = "20241104";
void *x = (void *) &A;
void *y = 2 + (void *) &B;
unsigned short P = *(unsigned short *) x;
unsigned short Q = *(unsigned short *) y;
printf("0x%04x", (unsigned short) (P - Q));
```

提示：‘0’，‘1’，…，‘9’的 ASCII 码分别是 0x30, 0x31, …, 0x39。

- A. 0xff05 B. 0x04ff C. 0x0822 D. 0x0800

习题试炼 8

2. 在 x86-64 机器上运行如下代码，输出是：

```
char A[12] = "19260817";
char B[12] = "20241104";
void *x = (void *) &A;
void *y = 2 + (void *) &B;
unsigned short P = *(unsigned short *) x;
unsigned short Q = *(unsigned short *) y;
printf("0x%04x", (unsigned short) (P - Q));
```

提示：‘0’，‘1’，…，‘9’的 ASCII 码分别是 0x30, 0x31, …, 0x39。

- A. 0xff05 B. 0x04ff C. 0x0822 D. 0x0800

答案：B。

习题试炼 9

3. 我们熟悉的标准浮点格式 float 共有 32 位，其阶码字段和小数字段分别为 $k=8$ 位和 $n=23$ 位。如果我们维持 float 的总位数不变，但阶码字段变为 $k'=10$ 位，则下列说法中，正确的是：
- A. 修改后能表示的实数值的数量更多了
 - B. 修改后能表示的实数值的数量不变
 - C. 修改后能表示的实数值的数量更少了
 - D. 无法确定

习题试炼 9

答案: A。

修改后，阶码字段增加，小数字段减少，可以表示的 NaN 的数量减少，所以能表示的实数值增多。

习题试炼 10

1. 在 x86-64 机器上, 产生随机的 32 位有符号整数 x 和 y, 则下列表达式中不恒为真的是:

- A. $((x >> 1) << 1) \leq x$
- B. $((x | -x) >> 31) + 1 == !x$
- C. $(-x < y) == (-y < x)$
- D. $(x ^ y ^ ((\sim x) - y)) == (y ^ x ^ ((\sim y) - x))$

习题试炼 10

1. 在 x86-64 机器上，产生随机的 32 位有符号整数 x 和 y，则下列表达式中不恒为真的是：

- A. $((x >> 1) << 1) \leq x$
- B. $((x | -x) >> 31) + 1 == !x$
- C. $(-x < y) == (-y < x)$
- D. $(x \wedge y \wedge ((\sim x) - y)) == (y \wedge x \wedge ((\sim y) - x))$

答案：C，一个反例是 $x = T_{min}$, $y = 0$ 。

THANKS

Made by WEB-05

webrun@stu.pku.edu.cn

Reference: [WalkerCH]'s and [Arthals]'s presentations.



扫一扫上面的二维码图案，加我为朋友。