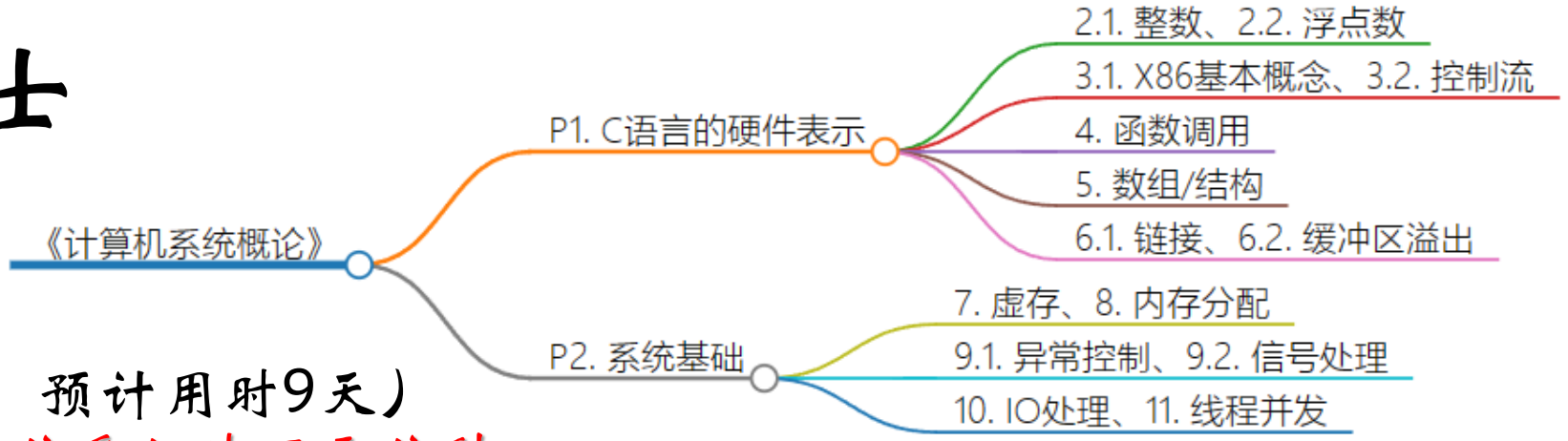
期末复习(1)

《计算机系统概论》习题课

张宇轩
yuxuanzh23@mails.tsinghua.edu.cn

一些复习小贴士



0. 尽早开始复习（9个分支，预计用时9天）
   合理规划复习时间，可以偏爱但请不要偏科

1. 整理手上有的复习材料，按课件顺序复习
   - 学习资源共享库：
     THUanonymous/bypass-thu-cst：有往年笔记可参考
     PKUanonym/REKCARC-TSC-UHT：只需看X86 MIPS不考！
   - CSAPP课后习题（必刷）：
     CSAPP-3e-Solutions (unofficial)
     汇编课以前作业
   - 参考解答＋习题课课件 易错题汇总
     尤其是页表、fork、I/O

2. 第8讲之后涉及许多库函数的使用
   一定要勤上机搞明白来

3. 3次实验内容也有可能会在考试中出现

课程	考试时间
计算机系统概论	240108（一） 晚上
数据结构	240109（二） 下午
形式语言与自动机	240113（六） 晚上
大学物理B(2)	240114（日） 上午
复变函数引论	240119（五） 下午
概率论与数理统计	240116（二） 上午

作业一 · Q2. 9位浮点数表示

见课件《2.2浮点数》P.10,11

Q2.2. 9位浮点数表示

注意各域位宽

假设存在一种符合IEEE浮点数标准的9位浮点数，由1个符号位、4位阶码、4位尾数组成，数值表示仍遵循$V = (-1)^s \cdot M \cdot 2^E$。请在下表中填空：

注意进制(写错0分)

IEEE 754标准

描述	9位二进制表示	M（十进制表示）	E（十进制表示）	V（十进制表示）
3.5				3.5
大于0的最小浮点数				

考察：规格化数
非规格化数

$\text{Bias} = 2^{e-1} - 1 = 2^{4-1} - 1 = 7$

- 3.5（规格化数）：
  $3.5_{10} = 11.1_2 = 1.11_2 \times 2^1$，于是$E = 1$，$M = 1.11_2 = 1.75_{10}$
- 大于0的最小浮点数（非规格化数）：
  其二进制表示为 0 0000 0001，那么$M = 0.0001_2 = 2^{-4} = 1/16$，$E = 1 - \text{Bias} = -6$，值等于$2^{-4} \times 2^{-6} = 2^{-10} = 1/1024$

描述	9位二进制表示	M（十进制表示）	E（十进制表示）	V（十进制表示）
3.5	0 1000 1100	1.75	1	3.5
大于0的最小浮点数	0 0000 0001	1/16	−6	1/1024

允许使用分数

作业一 · Q3.1. 加法溢出检测

要求通过位运算来为实现 `INT_MIN`，并赋值给 `MY_INT_MIN`。而 `INT_MIN` 的特征为除符号位为 `1`，其余位为 `0`，一般实现方式如下：

INT_MAX　　sign为0，其余为1

```
/* 表达式(1) */
// (1) 仅限于32位整数
const unsigned MY_INT_MIN = 1u << 31;
// (2) 通用实现，但涉及一次乘法运算（O2能优化掉）
const unsigned MY_INT_MIN = 1u << (sizeof(unsigned) * 8 - 1);
```

加法溢出发生于两个同符号整数相加后，得到的结果与被加数异号，如**负数加负数得到正数**。最直接的做法是判断结果的符号位与被加数 `si1` 或 `si2`（因为它们同号）的符号位是否相同。对此我们可以使用异或运算`来实现：
在C语言中，`x ^ y` 表示对 `x` 和 `y` 的每一位进行异或运算，自然包括了作为最高位的符号位：

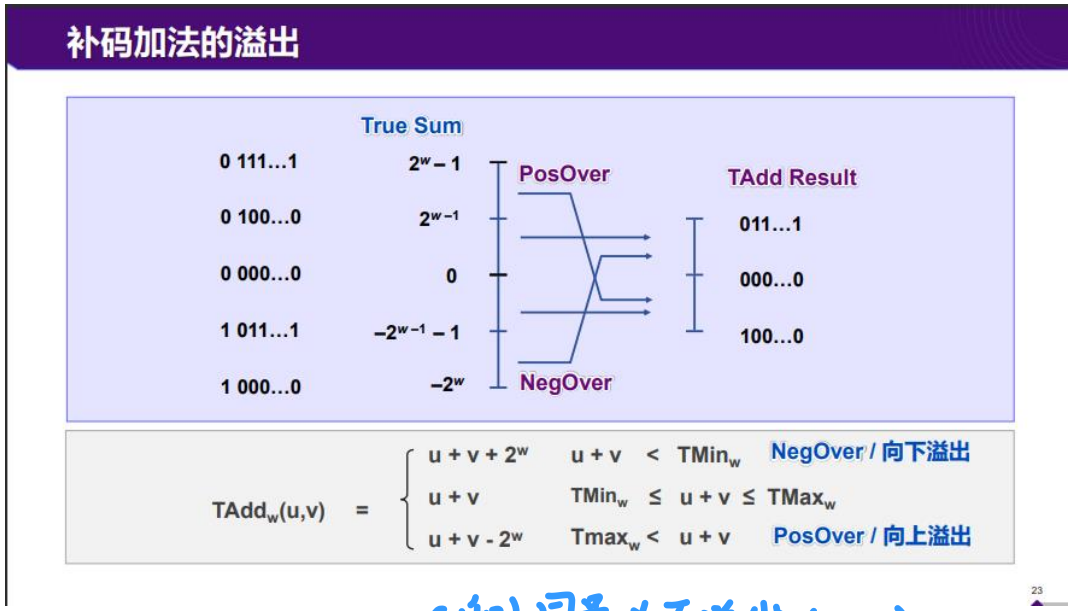
```
/* 表达式(2) */
// (1) 参考解
unsigned result = usum ^ si1;
// (2) 另一种解答，保险但没必要
unsigned result = (usum ^ si1) & (usum ^ si2);
// (3) 如果被加数是正数，直接看usum符号位；反之是负数，usum是正数，就将其符号位取反
unsigned result = (si1 > 0) ? usum : ~usum; // -usum也行
```

可是我们最终只想看符号位的值：假设加法实际没有溢出，但是除符号位外的位非全 `0`（比如 `0x01234567`），在C语言的 `if ()` 中，所有非零值对应逻辑值为 `true`，则该溢出判断成立——出bug了！

还记得一开始准备的 `MY_INT_MIN` 吗？它只有符号位为 `1`，其余位为 `0`："与"上 `MY_INT_MIN`，保留符号位的同时置其余位为 `0`，这时的结果是否非零完全取决于符号位，符合我们的预期。

```
if (result & MY_INT_MIN) ... // 符号位异号，发生溢出
```

见课件《2.1整数》P.23



a和b同号必不溢出(why?)

思考：如何实现减法溢出检测呢？

a 和 b 异号 ⇒ a − b = a + (−b)　　同号

Step 1　　Step 2

怎么修改呢？

作业二 · Q1.1. 匹配

汇编代码	对应函数 (填写函数名即可)	说明

foo2:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
testl %eax, %eax
jge .L2
addl $15, %eax
.L2:
sarl $4, %eax
movl %ebp, %esp
popl %ebp
ret
	choice5	`int choice5(int x) { return x / 16; }`
		通过 testl 指令设置 ZF 和 SF （并清空 CF，OF），
		若 SF 为 0 （表示正数），则直接跳转到 .L2 处执行 >> 4，
		否则先 + 15 再继续 这等价于向零舍入的带符号整数除法 **（见课件《2.1整数》P.28）**

foo3:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
shrl $31, %eax
movl %ebp, %esp
popl %ebp
ret
	choice1	`int choice1(int x) { return (x < 0); }`
		关键的地方是逻辑右移指令 shrl，在C语言中对于 x >> 4，
		若 x 是 int 则进行算术右移 **（见 课件《2.1整数》P.27）**，
		若 x 是 unsigned 才进行逻辑右移：这里的参数是 int x。
		因此最合适的是 choice1 返回符号位（最高位）的值
		（0 或 1）。

见课件《2.1整数》P.28

考察：见课件《2.1整数》P.25,27

（无/带符号整数与逻辑/算术右移）

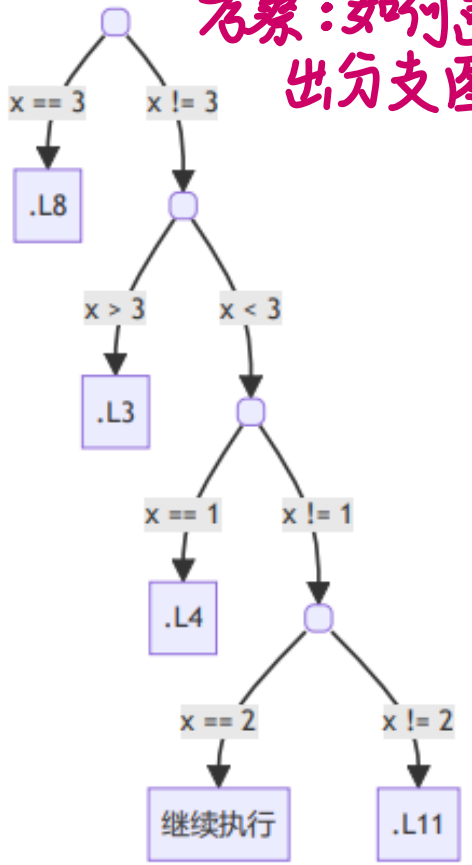
shrl sarl

作业二 · Q1.3. switch语句

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $3, %rdi
    je  .L8
    jg  .L3
    cmpq    $1, %rdi
    je  .L4
    cmpq    $2, %rdi
    jne .L11
    movq    %rsi, %rax
    cqto
    idivq   %rcx
    addq    %rcx, %rax
    ret
.L3:
    movl    $1, %eax
    subq    $5, %rdi
    subq    %rdx, %rax
    cmpq    $2, %rdi
    movq    %rax, %rcx
    movl    $2, %eax
    cmovb   %rcx, %rax
    ret
.L8:
    movl    $1, %eax
    addq    %rcx, %rax
    ret
.L4:
    movq    %rdx, %rax
    imulq   %rsi, %rax
    ret
.L11:
    movl    $2, %eax
    ret
```

```
long switch_eg(long x, long y, long z) {
    long w = 1;
    switch (x) {
        // TODO: 实现switch
    }
    return w;
}
```

考察：如何画出分支图



需要注意**在 `switch` 之前已经执行了 `int w = 1;`**。这时分情况讨论：

- `x == 3`：跳转到 `.L8`
  - 先执行 `w = 1;`，再执行 `w = w + z;` 后就返回了；
  - `w += z; break;`
- `x > 3`：跳转到 `.L3`
  - 先执行 `x -= 5;`（实际没有修改 `x` 的值），在赋值 `%rax` 前令 `%rcx = w - z`，`%rax = 2`；执行 `cmpq $2, %rdi`，来判断 `(x - 5) < 2` 是否为真（看 `(x - 5) - 2` 是否需要借位，若需要则置 `CF = 1`，反之为 `0`）；若成立（对应于 `CF = 1`）则执行 `cmovb` 指令，令 `%rax = %rcx`，否则保持 `%rax`；
  - 注意，在判断时不会考虑是否为有符号数：`x = 4` 时，`x - 5 = 0xffffffff > 2`，`x > 6` 时，`x - 5 >= 2`，它们对应的赋值为 `w = 2;`，**注意只有 `default` 分支有立即数赋值**，因此对应于 `default` 分支；`x - 5 < 2`，只有可能是 `x = 5` 和 `x = 6`，对应的赋值为 `w = w - z;`，执行完毕后返回；

```
case 5:
case 6:
    w -= z;
    break;
// 其他情况与default一致
```

尽量复用！

```
switch (x) {
    case 1:
        w = y * z; break;
    case 2:
        w = y / z;
    case 3:
        w += z; break;
    case 5:
    case 6:
        w -= z; break;
    default:
        w = 2; break;
}
```

- `x == 1`：跳转到 `.L4`
  - 先执行 `w = z`，再执行 `w = w * y`，之后返回；
  - `w = z * y; break;`
- `x == 2`：继续执行
  - 先执行 `w = y`，再执行 `w = w / z`（对应于 `cqto` 和 `idivq %rcx` 指令），最后执行 `w = w + z`
  - `w = y / z; w += z; break;`
  - 考虑到 `w += z; break;` 其实对应于 `x == 3`，可以按Fall through来写：

```
case 2:
    w = y / z;
case 3:
    w += z;
    break;
```

作业二 · Q2.1. 整数&浮点数回顾

课件《2.1整数》P.31 参考解答

Integer Puzzles

判断以下的推断或者等式是否成立 (不成立则给出示例)

- x，y 为32位带符号整数
- ux，uy为与x，y具有相同二进制表示的32位无符号整数

- x < 0 ⇒ ((x*2) < 0) 否，Tmin会下溢出
- ux >=0 是
- x & 7 == 7 ⇒ ((x<<30) < 0 是，最低三位为1
- ux >-1 否，比较时-1为Umax
- x > y ⇒ -x < -y 否，x=0, y=Tmin时
- x > 0 && y > 0 ⇒ x + y > 0 否，上溢出，x,y=Tmax
- x >= 0 ⇒ -x <= 0 是，-Tmax是Tmin-1
- x <= 0 ⇒ -x >= 0 否，-Tmin还是Tmin
- (x|-x) >> 31 == -1 否，x=0
- ux >> 3 == ux/8 是
- x >> 3 == x/8 否，不是向0舍入
- x & (x-1) != 0 否，x=0,1

以上仅从带符号整数、无符号数的定义出发，不涉及C语言的编译器具体实现，有兴趣同学可以深入参考C语言的“Undefined Behavior”

作业二 · Q2.1. 整数&浮点数回顾

课件《2.2浮点数》P.30 参考解答

Floating Point Puzzles

以下判断是否成立，如不成立请给出反例

```
int x = ...;
float f = ...;
double d = ...;
```

假设d 与 f 都不是 NaN

frac域隐含了1.，只保留了小数，所以尾数宽度为23+1=24位

- x == (int)(float) x 否，24位尾数，精度丢失
- x == (int)(double) x 是，53位尾数，精度保证
- f == (float)(double) f 是，不影响
- d == (float) d 否，精度减小
- f == -(-f); 是，仅改变符号位
- 2/3 == 2/3.0 否，C/C++下左式为0
- d < 0.0 → ((d*2) < 0.0) 是，符号位不影响
- d > f → -f > -d 是
- d * d >= 0.0 是
- (d+f)-d == f 否，(d+f)提升了f的精度，结果不一定完全一致

验证浮点数a, b一致，即验证 (a-b) < EPS 成立
EPS是浮点数的机器精度，2^(-p)，其中p是尾数宽度
如float，p=24，EPS=2^(-24)≈ 10^(-8)
（有兴趣者可以预习数值分析C01的内容）

作业三 · Q1.2. bfloat16

见课件《2.2浮点数》P.10,11


```
union {
    bfloat16 f;
    unsigned short s;
}
```

现在为 `f` 赋予 `bfloat16` 所能表示的最接近于 `1` 且大于 `1` 的数，在X86（小端序）机器上运行时，`s` 的十六进制值为多少?

再次强调!!

`bfloat16` 是由Google提出的一种半精度浮点数，`exp` 域为8位，`frac` 域为7位，`sign` 域为1位。除了位宽度差别外，`bfloat16` 的其它规格符合IEEE 754标准。

符合IEEE 754标准，即 `bfloat16` 严格遵循$(-1)^s \times M \times 2^E$的标准，且有$E = \exp - \text{bias} \Rightarrow \exp = E + \text{bias}$

考察：规格化浮点数

接近于1且大于1，说明这个数：

- 是一个规格化浮点数 ⇒ 尾数省略前导 `1` 、$\text{bias} = 2^{E-1} - 1$
- 正数，符号域$s = 0$，即 `s = 0b`
- **大于 1**，说明$M = 1.0000001$，省略前导 `1` ，即 `frac = 0000001b`
- $E = 0$，则$\text{bias} = 2^{8-1} - 1 = 127$，于是 `exp = 0b + 01111111b = 01111111b`

则二进制表示为 `0 01111111 0000001b = 0x3f81`

思考：

如果是最接近于0的非0数，s可能的值为？

不是严格>0

exp,frac域非全0

非规格化数 ⇒怎么表示？

↓ hint：正负0

有几个这样的数？