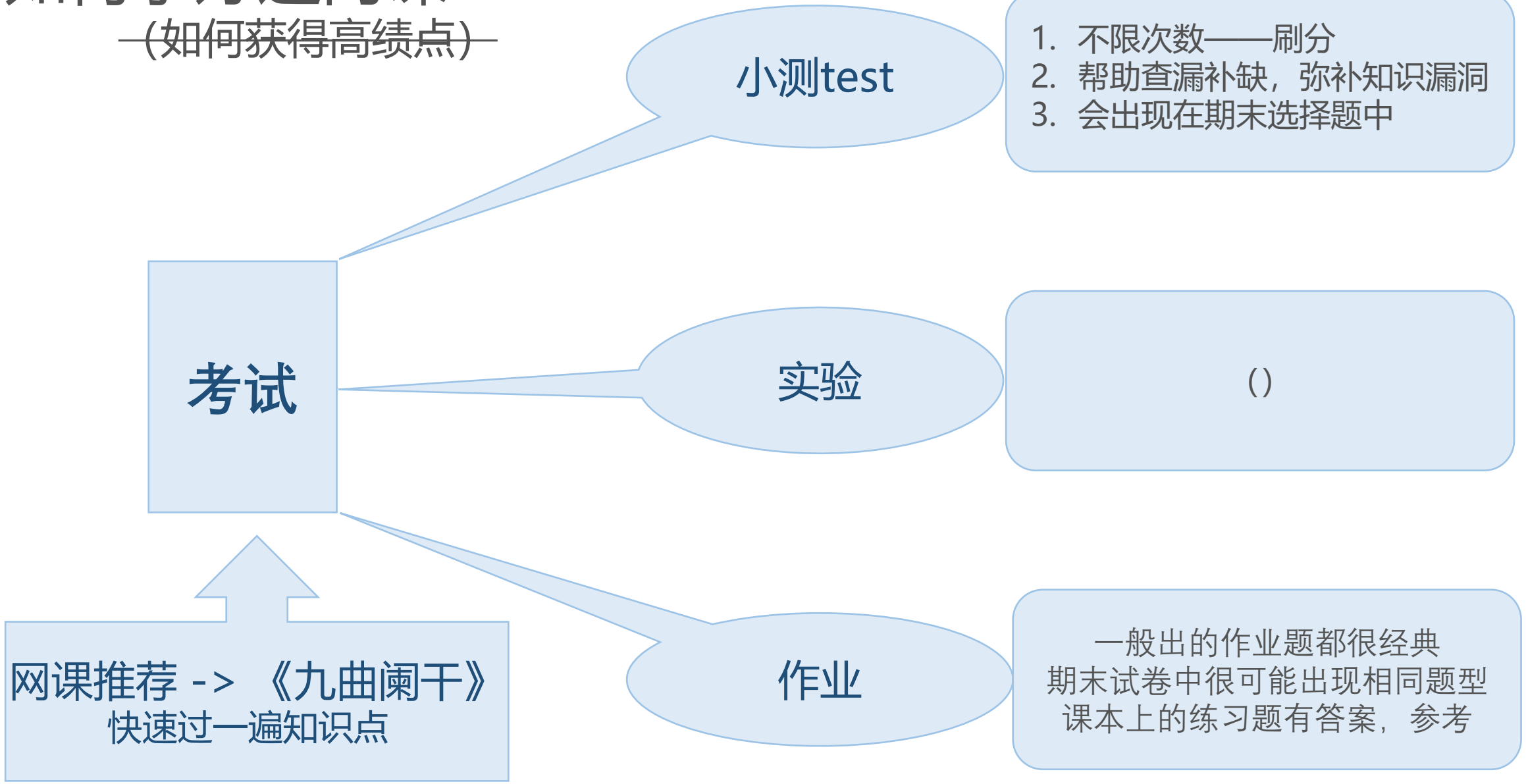
计算机系统/CSAPP

学业帮帮

—— 季子墨、李彤

如何学好这门课？？？

~~(如何获得高绩点)~~

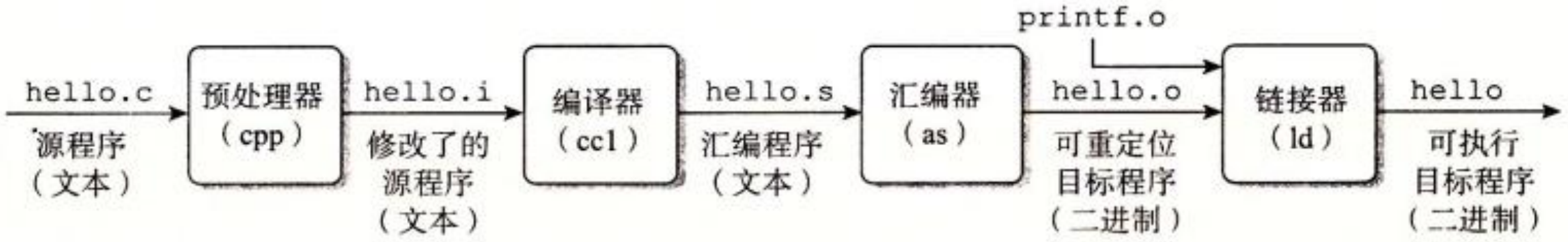


学业帮帮-CSAPP

第1题： 你希望能在学业帮帮上学到什么 [多选题]

选项	小计	比例
Lab讲解	16	66.67%
知识点梳理巩固	22	91.67%
难点专题讲解	18	75%
考前复习	24	100%
课外拓展学习	2	8.33%
其他	1	4.17%
本题有效填写人次	**24**	

编译过程：



- **预处理阶段：**.c -> .i;
- **编译阶段：**.i -> .s;
- **汇编阶段：**.s -> .o，得到可重定位程序，但是还不能直接运行;
- **链接阶段：**得到可执行文件;

系统的硬件组成：

- 总线：传递的是字（word）
- 主存：
  - 临时存储设备，
  - 由DRAM组成，
  - 是线性的字节数组，
  - 每个字节由唯一地址，
  - 这些地址从0开始；
- 处理器/CPU：执行指令的单元

1字节 = 8 位/bit

1字 = 4 字节

字节：最小可寻址的内存单位

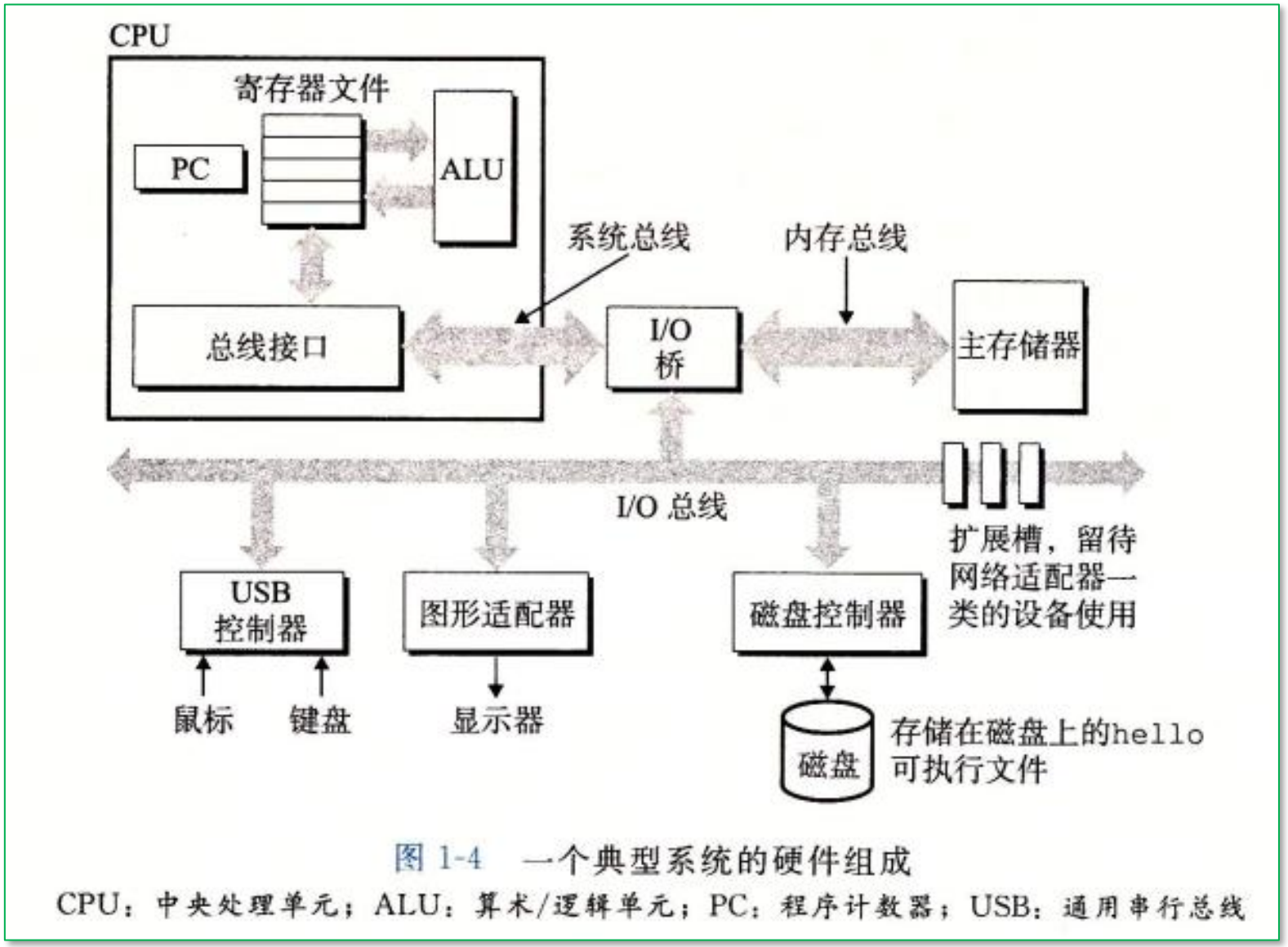
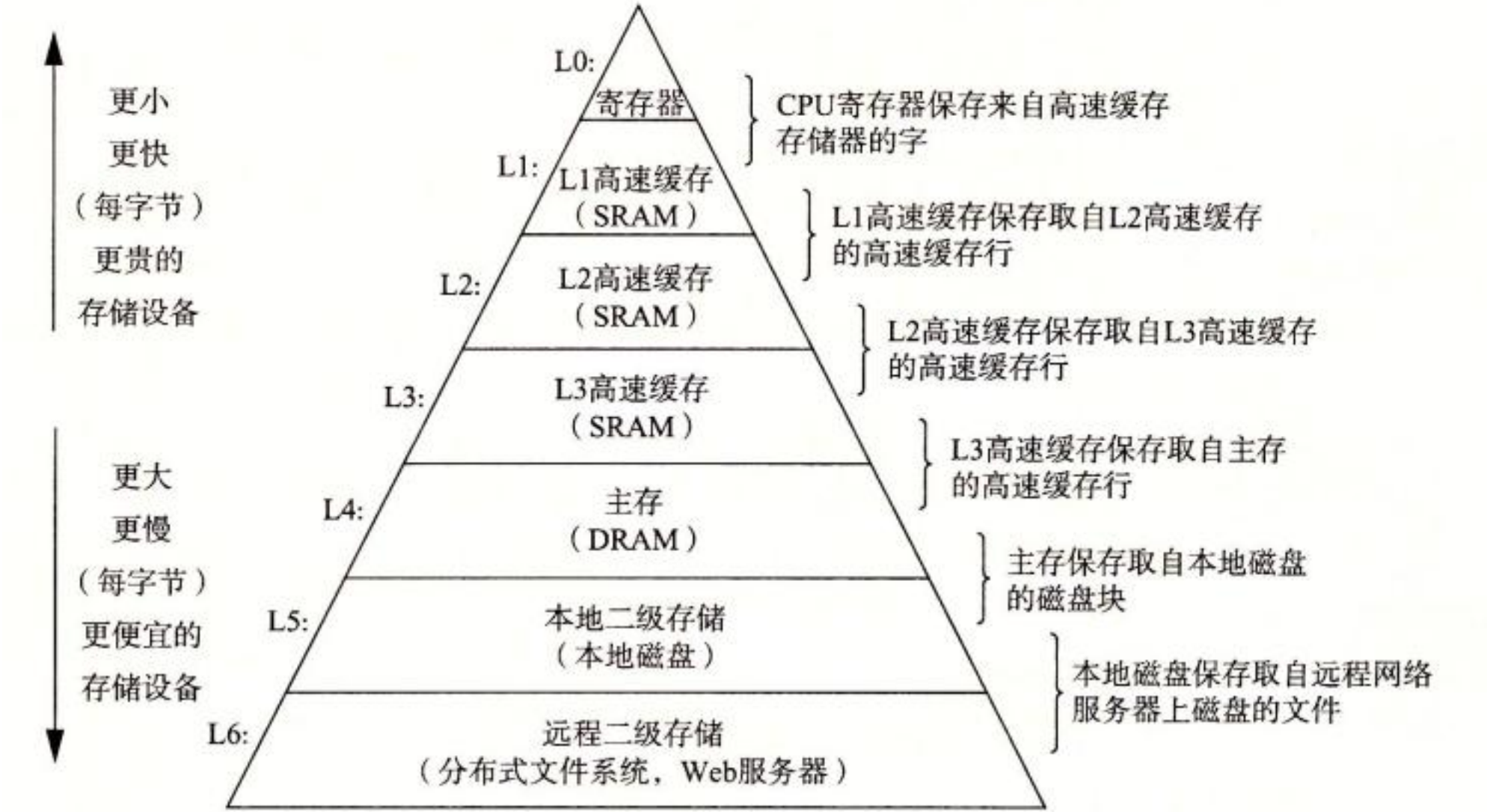


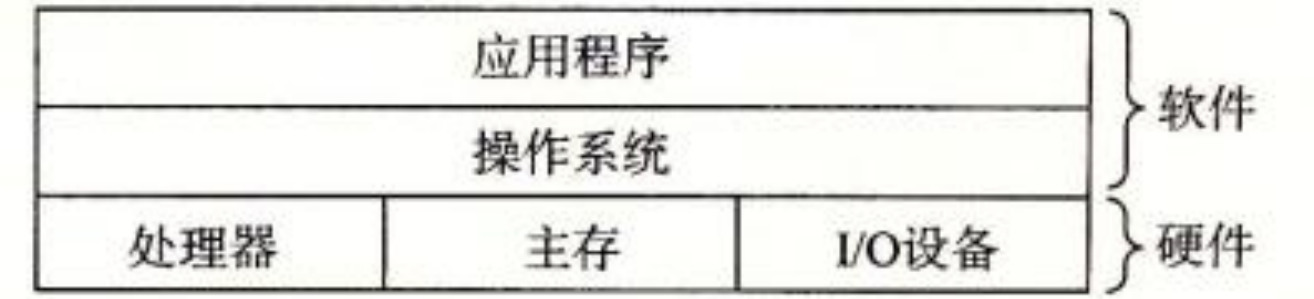
图 1-4　一个典型系统的硬件组成

CPU：中央处理单元；ALU：算术/逻辑单元；PC：程序计数器；USB：通用串行总线

存储层次结构：


更小
更快
（每字节）
更贵的
存储设备
更大
更慢
（每字节）
更便宜的
存储设备
L0:
寄存器
CPU寄存器保存来自高速缓存存储器的字
L1:
L1高速缓存（SRAM）
L1高速缓存保存取自L2高速缓存的高速缓存行
L2:
L2高速缓存（SRAM）
L2高速缓存保存取自L3高速缓存的高速缓存行
L3:
L3高速缓存（SRAM）
L3高速缓存保存取自主存的高速缓存行
L4:
主存（DRAM）
主存保存取自本地磁盘的磁盘块
L5:
本地二级存储（本地磁盘）
本地磁盘保存取自远程网络服务器上磁盘的文件
L6:
远程二级存储（分布式文件系统，Web服务器）

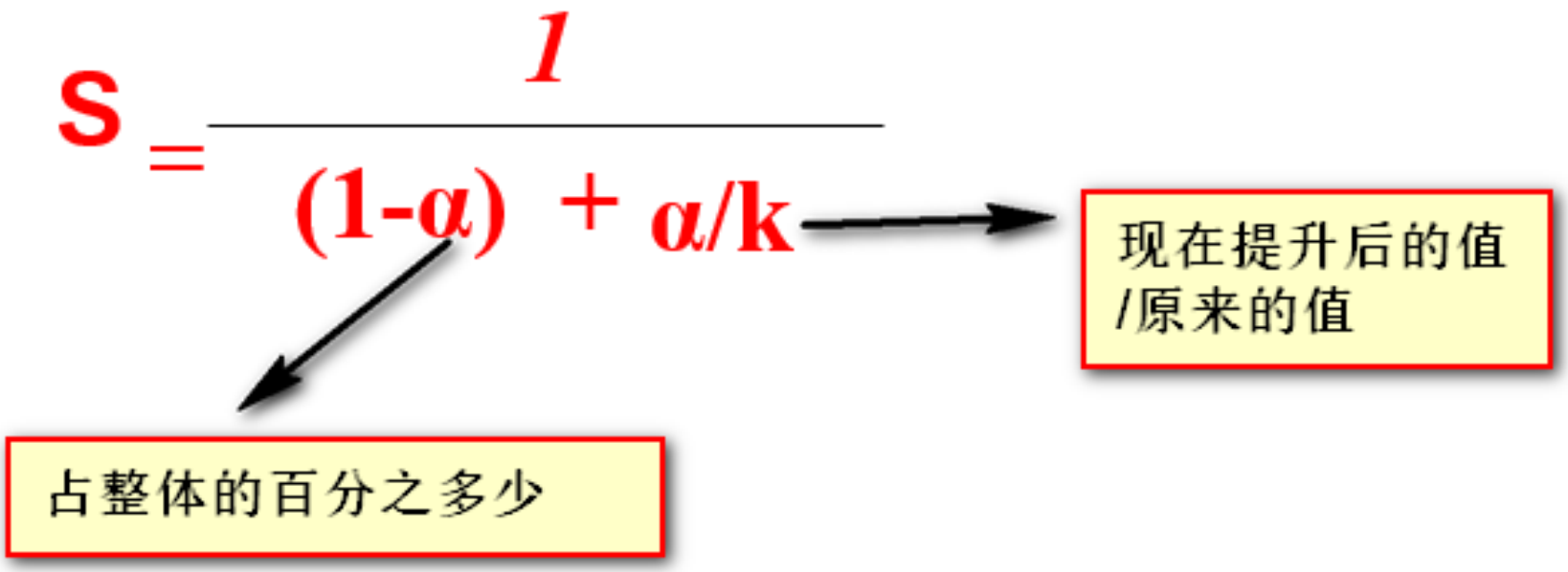
计算机系统的抽象表示：



考点：谁是谁的抽象？

概念	意义
进程	程序（静态）运行起来就是进程，是动态的
线程	比进程更小的运行单元，一个进程可能由多个线程组成
并发	不同的进程交替运行，但是不能同时运行
并行	不同的进程同时运行
虚拟内存	给进程提供的独立地址空间的假象
单处理器	一个芯片只有一个CPU，因此只能并发执行指令
多处理器	一个芯片中集成多个CPU，所以可以并行执行指令

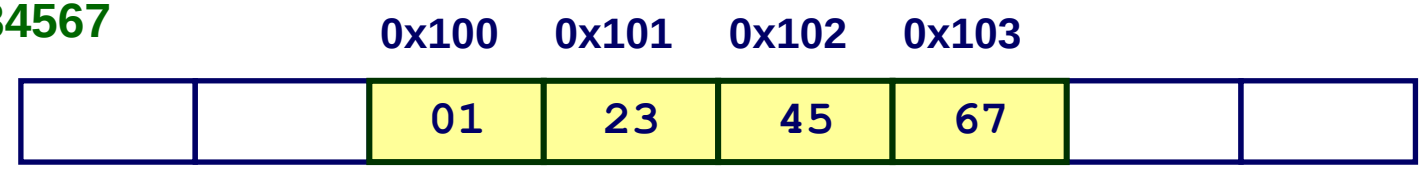
Amdahl 定律



- S：加速比
- α：某部分所需执行时间与整个应用程序执行所需时间的比例
- k：该部分性能提升的比例

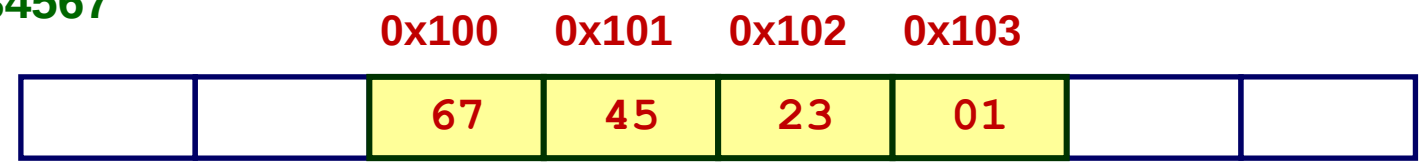
大端法：最高有效字节在最低地址（即前面）

int a = 0x01234567



小端法：最低有效字节在最低地址

int a = 0x01234567



只有对字节进行操作的时候才会有差别

练习题 2.5 思考下面对 show_bytes 的三次调用：

```
int val = 0x87654321;
byte_pointer valp = (byte_pointer) &val;
show_bytes(valp, 1); /* A. */
show_bytes(valp, 2); /* B. */
show_bytes(valp, 3); /* C. */
```

指出在小端法机器和大端法机器上，每次调用的输出值。

A. 小端法：　　大端法：

B. 小端法：　　大端法：

C. 小端法：　　大端法：

文本数据比二进制数据具有更强的平台独立性

布尔代数：

- 布尔运算没有进位
- 使用布尔运算实现swap（必看）

步骤	*x	*y
初始	a	b
步骤1	a	a^ b
步骤2	a ^ (a ^ b) = (a ^ a) ^ b =b	a^ b
步骤3	b	b^ (a ^ b) = (b ^ b) ^ a = a

- 注意区分逻辑运算和按位运算
- 逻辑运算遵循短电路原则
- 移位主要区分右移时是逻辑右移还是算术右移（无符号数只能逻辑右移）
- 当一个数只有m位时，那么移位的距离实际为 k mod m 位

整数表示：

C数据类型	最小值	最大值
[signed]char	-127	127
unsigned char	0	255
short	-32 767	32 767
unsigned short	0	65 535
int	-32 767	32 767
unsigned	0	65 535
long	-2 147 483 647	2 147 483 647
unsigned long	0	4 294 967 295
int32_t	-2 147 483 648	2 147 483 647
uint32_t	0	4 294 967 295
int64_t	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
uint64_t	0	18 446 744 073 709 551 615

图 2-11　C 语言的整型数据类型的保证的取值范围。C 语言标准要求这些数据类型必须至少具有这样的取值范围

- 强制类型转换的结果保持位值不变，只是改变了解释这些位向量的方式
- 有符号和无符号之间的转化
- 当表达式中 unsigned 和 signed 混用时，signed 将被强制转换为 unsigned

扩展与截断

- 扩展：
  - 无符号数：零扩展
  - 补码数:符号扩展
- 截断：
  - 即取模运算，只保留低位值
  - 补码数截断需要将最高位转为符号位

整数运算

- 无符号加法：
  - res = x + y / （x + y）mod 2m
- 补码加法：
  - res = x + y / （x + y）mod 2m
- 无符号的非：
  - res = 2m - x
- 补码的非：$-_w^t x = \begin{cases} TMin_w, & x = TMin_w \\ -x, & x > TMin_w \end{cases}$
- 无符号乘法：
  - res = （x * y）mod 2m
- 补码数乘法：低位的位模式同无符号乘法结果相同
- 乘以常数：用移位和加法代替乘法，移位因子转为2的幂（移位记得加括号）

溢出检测？

整数运算（续

- 除以2的幂：
  - 无符号逻辑右移，补码数算数右移
  - 当补码数为负数时，需要加上一个偏置值（bias）从而实现向零舍入
  - bias = $2^k - 1$，k表示补码要移动的位数

浮点数 $(-1)^s * M * 2^E$

- s（sign）/符号位：
  - 决定这数是负数（s = 1）还是正数（s = 0），而对于数值0的符号位解释作为特殊情况处理
- M/尾数：
  - 范围是$1-2^{-e}$，或者是$0-1^{-e}$
- E/阶码：
  - 对浮点数加权，权重是2的E次幂(可能是负数)

单精度浮点数

$(-1)^S * M * 2^E$

双精度浮点数

- 非规格化数：exp全为0
- 无穷大：exp全为1，frac全为0
- NaN：exp全为1，frac不为0

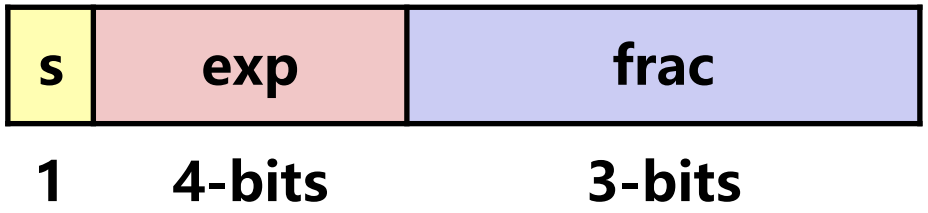
exp和阶码、M和frac的关系/区别？

规格化数 VS 非规格化数：

- **规格化数：** M = 1 + frac，E = e - Bias
- 非规格化数：M = frac，E = 1-Bias

Bias = $2^{k-1} - 1$，其中 k 是 exp 的位数

一些特殊值：



8位浮点格式，4位阶码 3位尾数，

意义	二进制	十进制
最小的非规格化数	0 0000 000	0
最大的非规格化数	0 0000 111	7 / 512
最小的规格化数	0 0001 000	8 / 512 或 1 / 64
最大的规格化数	0 1110 111	1920 / 8 或 240
无穷大	1 1111 000 / 0 1111 000	+∞ 或 -∞

舍入

- 向下舍入
- 向上舍入
- 向零舍入
- 向偶数舍入：使得结果的最低有效位是偶数。

只有对两个数的中间值才使用向偶数舍入，非中间值直接四舍五入即可

- 例：向偶数舍入到小数点后第二位

1.2349999	1.23	(小于中间值)
1.2350001	1.24	(大于中间值)
1.2350000	1.24	(中间，向上舍入)
1.2450000	1.24	(中间，向下舍入)

浮点数运算

- 加法：
  - 阶码对齐，尾数相加
  - 可交换，满足单调性，不满足结合律
- 乘法：
  - 符号位异或，阶码（指数）相加，尾数相乘
  - 可交换，满足单调性，不满足结合律和分配律

C语言中强制转换原则

转换前	转换后	结果
int	float	不会溢出，可能被舍入
float / double	int	向零舍入，值可能溢出
int / float	double	不会溢出
double	float	溢出+∞或-∞，可能被舍入

C声明	Intel数据类型	汇编代码后缀	大小(字节)
char	字节	b	1
short	字	w	2
int	双字	l	4
long	四字	q	8
char*	四字	q	8
float	单精度	s	4
double	双精度	l	8

图 3-1　C语言数据类型在 x86-64 中的大小。在 64 位机器中，指针长 8 字节

图 3-2 整数寄存器。所有 16 个寄存器的低位部分都可以作为字节、字(16 位)、双字(32 位)和四字(64 位)数字来访问

- 生成 1 字节和 2 字节数字的指令会保持剩下的字节不变
- 生成 4 字节数字的指令会把高位 4 个字节置为 0

- 操作数类型：
  - **立即数：**
    - 以$开头，如 $0xff
  - **寄存器：**
    - Ra表示任意寄存器，[Ra]表示任意寄存器中的值
  - **内存引用：**
    - 根据计算出的地址访问某个内存位置
    - 常见寻址方式：Imm(rb, ri, s) = Imm + R[rb] + R[ri] * s
      - 其中s必须是1、2、4或8

- r_b：基址寄存器
- r_i：变址寄存器
- s ：比例因子
- Imm：立即数偏移
- r_b 基址和 r_i 变址寄存器都必须是 64 位寄存器

类型	格式	操作数值	名称
立即数	$\$Imm$	Imm	立即数寻址
寄存器	r_a	$R[r_a]$	寄存器寻址
存储器	Imm	$M[Imm]$	绝对寻址
存储器	(r_a)	$M[R[r_a]]$	间接寻址
存储器	$Imm(r_b)$	$M[Imm+R[r_b]]$	（基址＋偏移量）寻址
存储器	(r_b, r_i)	$M[R[r_b]+R[r_i]]$	变址寻址
存储器	$Imm(r_b, r_i)$	$M[Imm+R[r_b]+R[r_i]]$	变址寻址
存储器	$(,r_i, s)$	$M[R[r_i] \cdot s]$	比例变址寻址
存储器	$Imm(,r_i,s)$	$M[Imm+R[r_i] \cdot s]$	比例变址寻址
存储器	(r_b, r_i, s)	$M[R[r_b]+R[r_i] \cdot s]$	比例变址寻址
存储器	$Imm(r_b, r_i, s)$	$M[Imm+R[r_b]+R[r_i] \cdot s]$	比例变址寻址

图 3-3　操作数格式。操作数可以表示立即数(常数)值、寄存器值或是来自内存的值。比例因子 s 必须是 1、2、4 或者 8

MOV指令

指令	效果	描述
MOV S，D	D←S	传送
movb		传送字节
movw		传送字
movl		传送双字
movq		传送四字
movabsq I，R	R←I	传送绝对的四字

图 3-4　简单的数据传送指令

- 源操作数是立即数，存储在寄存器或存储器中
- 目的操作数只能是寄存器或内存地址
- 指令的两个操作数不能都指向内存地址
- movl以寄存器为目的时，会使寄存器的高32位置零

```
1    movabsq $0x0011223344556677, %rax    %rax = 0011223344556677
2    movb    $-1, %al                     %rax = 00112233445566FF
3    movw    $-1, %ax                     %rax = 001122334455FFFF
4    movl    $-1, %eax                    %rax = 00000000FFFFFFFF
5    movq    $-1, %rax                    %rax = FFFFFFFFFFFFFFFF
```

指令	效果	描述
pushq S	R[%rsp]←R[%rsp]−8; M[R[%rsp]]←S	将四字压入栈
popq D	D←M[R[%rsp]]; R[%rsp]←R[%rsp]+8	将四字弹出栈

图 3-8　入栈和出栈指令

- 栈顶：压入和删除数据的地方
- 栈往地址减小的方向生长

指令	效果	描述
`leaq` S, D	$D \leftarrow \&S$	加载有效地址
INC D	$D \leftarrow D + 1$	加1
DEC D	$D \leftarrow D - 1$	减1
NEG D	$D \leftarrow -D$	取负
NOT D	$D \leftarrow \sim D$	取补
ADD S, D	$D \leftarrow D + S$	加
SUB S, D	$D \leftarrow D - S$	减
IMUL S, D	$D \leftarrow D * S$	乘
XOR S, D	$D \leftarrow D \hat{} S$	异或
OR S, D	$D \leftarrow D \mid S$	或
AND S, D	$D \leftarrow D \,\&\, S$	与
SAL k, D	$D \leftarrow D << k$	左移
SHL k, D	$D \leftarrow D << k$	左移（等同于SAL）
SAR k, D	$D \leftarrow D >>_A k$	算术右移
SHR k, D	$D \leftarrow D >>_L k$	逻辑右移

leap指令

- 从内存读数据到寄存器，没有大小变种（无后缀）
- **le**aq Src，Dst 将 Src (内存)的有效地址写入到 Dst（寄存器）

```
  long scale(long x, long y, long z)
  x in %rdi, y in %rsi, z in %rdx
scale:
  leaq    (%rdi,%rsi,4), %rax     x + 4*y
  leaq    (%rdx,%rdx,2), %rdx     z + 2*z = 3*z
  leaq    (%rax,%rdx,4), %rax     (x+4*y) + 4*(3*z) = x + 4*y + 12*z
  ret
```

指令		基于	描述
CMP	S_1，S_2	S_2-S_1	比较
cmpb			比较字节
cmpw			比较字
cmpl			比较双字
cmpq			比较四字
TEST	S_1，S_2	$S_1 \& S_2$	测试
testb			测试字节
testw			测试字
testl			测试双字
testq			测试四字

图 3-13　比较和测试指令。这些指令不修改任何寄存器的值，只设置条件码

指令		同义名	效果	设置条件
sete	*D*	setz	*D* ← ZF	相等/零
setne	*D*	setnz	*D* ← ~ZF	不等/非零
sets	*D*		*D* ← SF	负数
setns	*D*		*D* ← ~SF	非负数
setg	*D*	setnle	*D* ← ~(SF ^ OF) & ~ZF	大于（有符号>）
setge	*D*	setnl	*D* ← ~(SF ^ OF)	大于等于（有符号>=）
setl	*D*	setnge	*D* ← SF ^ OF	小于（有符号<）
setle	*D*	setng	*D* ← (SF ^ OF) | ZF	小于等于（有符号<=）
seta	*D*	setnbe	*D* ← ~CF & ~ZF	超过（无符号>）
setae	*D*	setnb	*D* ← ~CF	超过或相等（无符号>=）
setb	*D*	setnae	*D* ← CF	低于（无符号<）
setbe	*D*	setna	*D* ← CF | ZF	低于或相等（无符号<=）

图 3-14　SET 指令。每条指令根据条件码的某种组合，将一个字节设置为 0 或者 1。有些指令有“同义名”，也就是同一条机器指令有别的名字

CF：进位标志。最近的操作使最高位产生了进位。可用来检查无符号操作的溢出。

ZF：零标志。最近的操作得出的结果为 0。

SF：符号标志。最近的操作得到的结果为负数。

OF：溢出标志。最近的操作导致一个补码溢出——正溢出或负溢出。

leap不改变任何条件码

PC相对寻址

下面是链接后的程序反汇编版本：

```
1    4004d0:  48 89 f8          mov    %rdi,%rax
2    4004d3:  eb 03             jmp    4004d8 <loop+0x8>
3    4004d5:  48 d1 f8          sar    %rax
4    4004d8:  48 85 c0          test   %rax,%rax
5    4004db:  7f f8             jg     4004d5 <loop+0x5>
6    4004dd:  f3 c3             repz retq
```

练习题 3.15 在下面这些反汇编二进制代码节选中，有些信息被 X 代替了。回答下列关于这些指令的问题。

A. 下面 je 指令的目标是什么？（在此，你不需要知道任何有关 callq 指令的信息。）

```
4003fa: 74 02          je     XXXXXX
4003fc: ff d0          callq  *%rax
```

0x4003fc + 0x02 = 0x4003fe

B. 下面 je 指令的目标是什么？

```
40042f: 74 f4          je     XXXXXX
400431: 5d             pop    %rbp
```

0x40042f + 0xf4
= 0x40042f – 12 = 0x400423

C. ja 和 pop 指令的地址是多少？

```
XXXXXX: 77 02          ja     400547
XXXXXX: 5d             pop    %rbp
```

0x400543
0x400545

0x400547 – 0x02 = 0x400545
0x400545 – 2 = 0x400543

D. 在下面的代码中，跳转目标的编码是 PC 相对的，且是一个 4 字节补码数。字节按照从最低位到最高位的顺序列出，反映出 x86-64 的小端法字节顺序。跳转目标的地址是什么？

```
4005e8: e9 73 ff ff ff     jmpq   XXXXXXX
4005ed: 90                 nop
```

0x4005ed + 0xffffff73
= 0x4005ed – 141 = 0x400560

do while 循环

```
    long fact_do(long n)
    n in %rdi
1   fact_do:
2     movl    $1, %eax        Set result = 1
3   .L2:                    loop:
4     imulq   %rdi, %rax      Compute result *= n
5     subq    $1, %rdi        Decrement n
6     cmpq    $1, %rdi        Compare n:1
7     jg      .L2             If >, goto loop
8     rep; ret                Return
```

c）对应的汇编代码

while 循环

```
    long fact_while(long n)
    n in %rdi
1   fact_while:
2     cmpq    $1, %rdi        Compare n:1
3     jle     .L7             If <=, goto done
4     movl    $1, %eax        Set result = 1
5   .L6:                    loop:
6     imulq   %rdi, %rax      Compute result *= n
7     subq    $1, %rdi        Decrement n
8     cmpq    $1, %rdi        Compare n:1
9     jne     .L6             If !=, goto loop
10    rep; ret                Return
11  .L7:                    done:
12    movl    $1, %eax        Compute result = 1
13    ret                     Return
```

Jguanprdteod-Mdiodd方e式