9.28 信息的表示和处理/程序的机器级表示（一）

李浩雨，施朱鸣

字节数

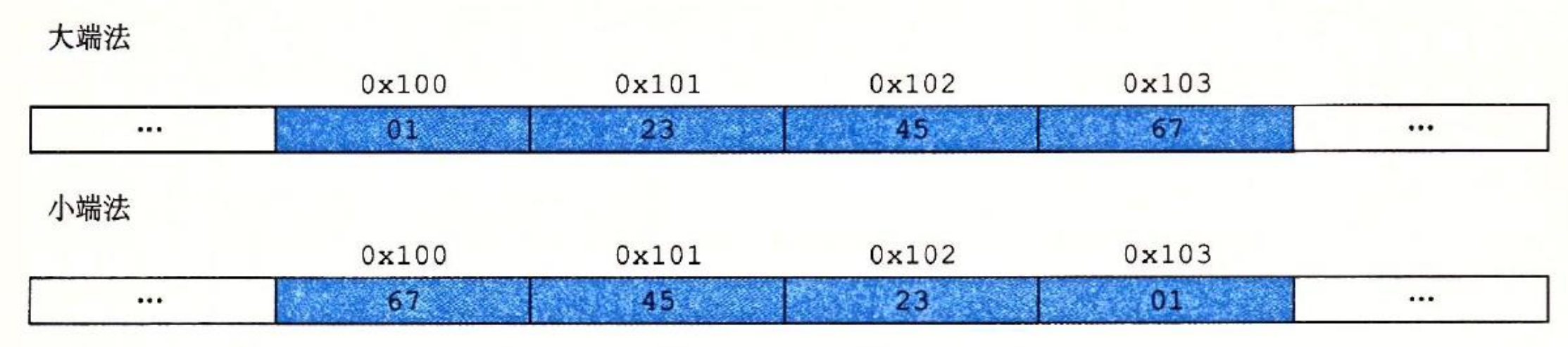
- *识记char, short, int, long long, float, double, pointer等类型的字节数，注意结合大小端考虑这些字节在内存中的存储*

- **32位机器和64位机器最大的区别在于指针大小**
- **不要犯晕，int是4字节，(x86-64下)int*是8字节**
- *对64位程序，gcc 4.6 以上版本可使用__int128_t，__uint128_t .但这并非c/c++ 标准
- *c/c++标准不保证char是有符号数

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
float	4	4	4
double	8	8	8
long double	-	-	10/16
pointer	4	4	8

大小端

- **小端法低地址放低字节**
- 不要记混，考试时可能把低地址画在右边，高地址画在左边
- ***看见往地址里填数字的问题要条件反射地考虑大小端***
- 典型大端： **Sun**, PPC Mac, **Internet,** 多数 **IBM,** 多数 **Oracle**
- 典型小端: **(intel)x86**, ARM processors running Android, iOS, and Windows



位运算，逻辑运算，移位

- &，|, ~，^.位运算优先级低于加减法

例：x^y^(~x)|y ＝＝ y^x^(~y)|x.

- 与非，或非的完备性（特别地，与或不完备）
- 异或满足交换律

- 对逻辑运算&&，||而言，如果第一个参数求出结果，就不会计算第二个参数
- int * p=NULL;
- if(p==NULL||*p==0){;}　　//OK

- 算术右移和逻辑右移
- *对int32_t而言，左移32位≠置零

整数

- 补码（two's complement），反码，原码

- 有符号数和无符号数，符号扩展和零扩展
- 注意强制类型转换中，有符号和无符号数运算时会转换为无符号
- 例：(!!x)-sizeof(short) > 0

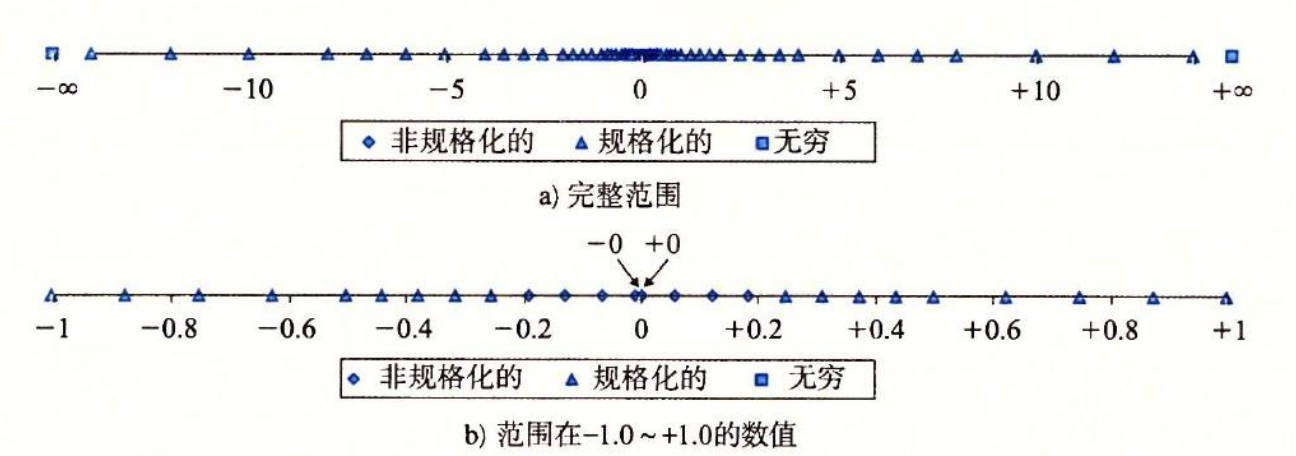
整数

- 溢出和截断
- 对于补码加法：正数+负数不会溢出；正数+正数不可能溢出为正数
- 事实上，如果开-O3优化，表达式 -(0x80000000)>-1 可能为真，这是因为编译器为我们做了我们不希望做的优化。一般地，我们默认采用-Og优化的结果。

- 算术运算
- 有符号数（无符号数）加法（乘法）具有位级等价性（用这一条加快解题速度）
- 整数除法总是舍入到0（不要与浮点“向偶数舍入”混淆）、
- 特别地，(x<0 ? x + (1<<k) – 1 : x)>>k会计算$x/2^k$

浮点数

- 符号位(s)：阶码位(k)：尾数位(n)
- 单精度 1:8:23；双精度1:11:52
- 熟练掌握规格化数，非规格化数，无穷大，非数的表示和比较（例如无穷大之间和非数之间的比较）；快速进行浮点数和十进制数的转化；能推导或提前识记特殊数值（如浮点可精确表示的最大整数，最大规格化数，最大非规格化数，最小非规格化数，最小非负非规格化数等）



浮点舍入

- 向最近的值舍入：那中间值怎么办？四种处理方式
- 向上舍入
- 向下舍入：算术右移>>
- 向0舍入：整数除法a/b
- 向偶数舍入：默认的
- 只有中间值（XXX.YYYY1000）$_2$才向“偶数”舍入
- 最低有效位倾向于0

浮点运算

- 乘法：分别计算s, M, E, 若$M \geqslant 2$ 则调整M 和E，最后处理溢出和舍入
- 加法：对齐后再计算, 若$M \geqslant 2$ 或$M < 1$ 则调整M 和E，最后处理溢出和舍入
- 可以交换，不可结合（扔掉小尾巴），不可分配（inf-inf=NaN）
- 除了NaN和inf,-inf，保留单调性
- 有NaN 且比较运算符不为!= 时返回false , 为!= 返回true
- NaN==NaN(false) , NaN!=NaN(true)
- 如何制备NaN?

汇编与反汇编

- > gcc –Og –S test.c 编译器制备test.s汇编代码
- > gcc –Og –c test.c 编译器制备test.s汇编代码，然后汇编器制备二进制目标代码文件test.o
- objdump –d test.o 反汇编器用.o文件制备汇编

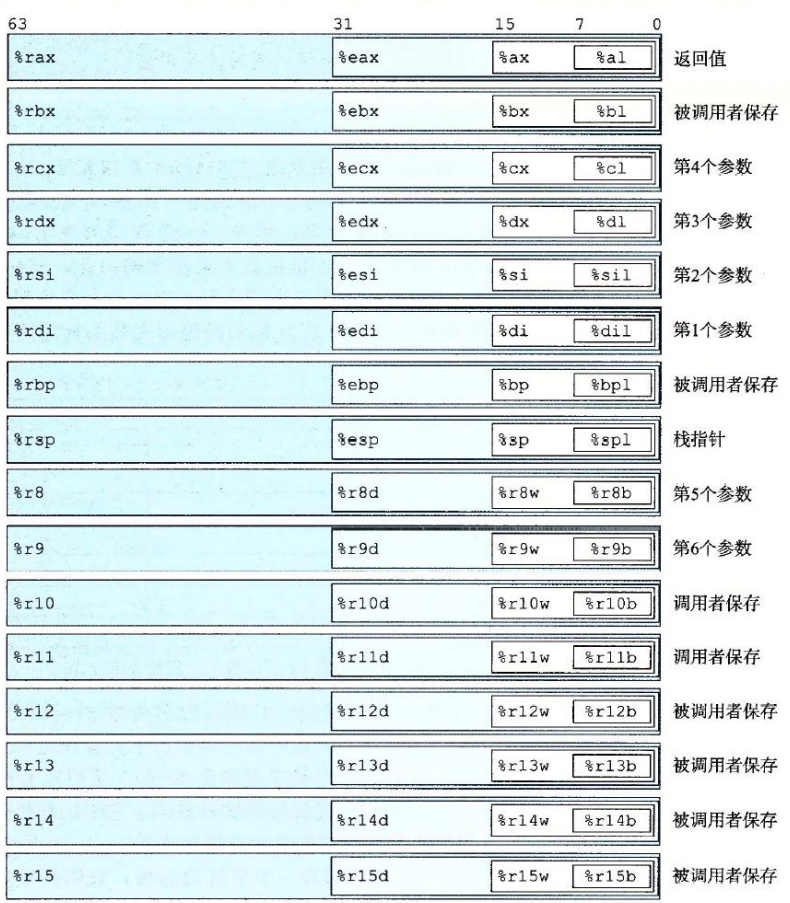
数据格式

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

寄存器

- 生成1~2byte的指令保持余下不变
- 生成4byte的指令把高4byte置0



操作数

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

数据传送指令

- 传送指令把一个**立即数**（直接给出、或者存在寄存器、或者内存里）传递到一个**位置**（寄存器或者内存里）
- 两个操作数不能同时是内存位置（可以用寄存器为媒介）
- movl以寄存器为目标时高4byte清零
- movq和movabsq的区别？movq只能以32位补码数字为源操作数，然后符号拓展，movabsq则以64位立即数位操作源数，但只能移到寄存器

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

数据传送指令

- 扩展规则：z(ero)和s(ign)
- 把4byte零扩展到8byte：movl到自己（高位清零）
- cltq：等价于movslq %eax,%rax
- leaq：只计算地址，不去对应地址取值，被拿来做简易计算
- 强制类型转换时既涉及到大小变化又有C语言符号变化时，先变大小

指令	效果	描述
MOVZ S, R	R←零扩展(S)	以零扩展进行传送
movzbw		将做了零扩展的字节传送到字
movzbl		将做了零扩展的字节传送到双字
movzwl		将做了零扩展的字传送到双字
movzbq		将做了零扩展的字节传送到四字
movzwq		将做了零扩展的字传送到四字

图 3-5 零扩展数据传送指令。这些指令以寄存器或内存地址作为源，以寄存器作为目的

指令	效果	描述
MOVS S, R	R←符号扩展(S)	传送符号扩展的字节
movsbw		将做了符号扩展的字节传送到字
movsbl		将做了符号扩展的字节传送到双字
movswl		将做了符号扩展的字传送到双字
movsbq		将做了符号扩展的字节传送到四字
movswq		将做了符号扩展的字传送到四字
movslq		将做了符号扩展的双字传送到四字
cltq	%rax ←符号扩展(%eax)	把%eax符号扩展到%rax

图 3-6 符号扩展数据传送指令。MOVS指令以寄存器或内存地址作为源，以寄存器作为目的。cltq指令只作用于寄存器%eax和%rax