**计算机系统基础期中复习——知识整理**

**第二章**

三部分：无符号数，补码，浮点数

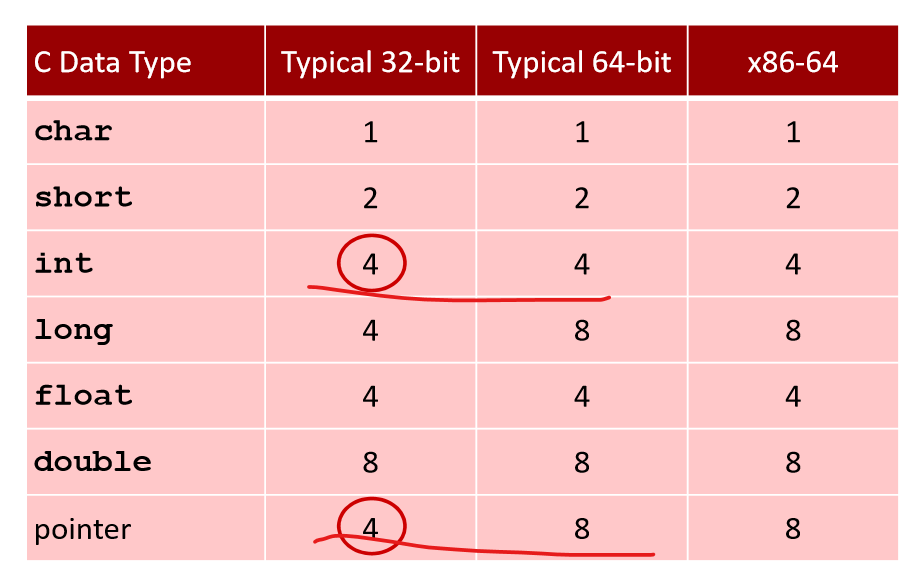
对比：·整数：数值范围较小，但精确

·浮点数：数值范围较大，但只是近似的

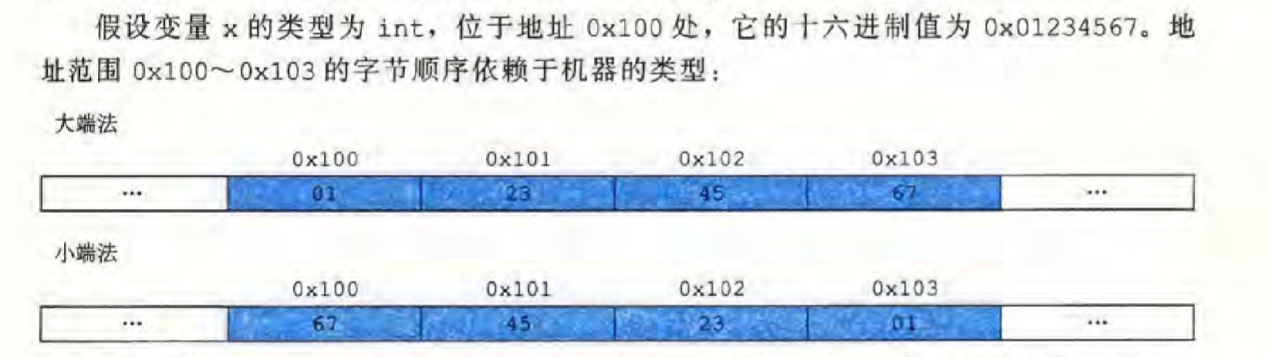
**2.1**

**·**字节：8位

对一个字长w位的机器而言，虚拟地址的范围为0~2^w-1，程序最多访问2^w个字节。



**·**大端法和小端法：



**·**布尔代数：

|  |  |  |
| --- | --- | --- |
| 布尔运算 | 逻辑运算 | 命题逻辑 |
| & | AND | ︿ |
| | | OR | ﹀ |

注意，^是异或

注意：位运算符和逻辑运算符的区别~和！，&和&&，|和||

逻辑运算符的特点：如果第一个参数值就能确定表达式结果，就不会对第二个参数求值。

**·**移位运算：

从左至右结合。

左移：右端补0

右移：·算数右移：左端补最高有效位

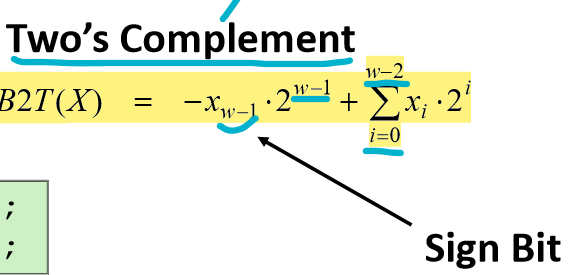
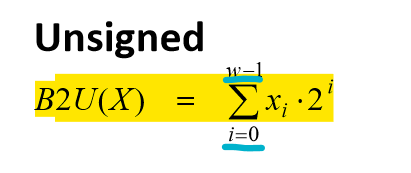
·逻辑右移：左端补0

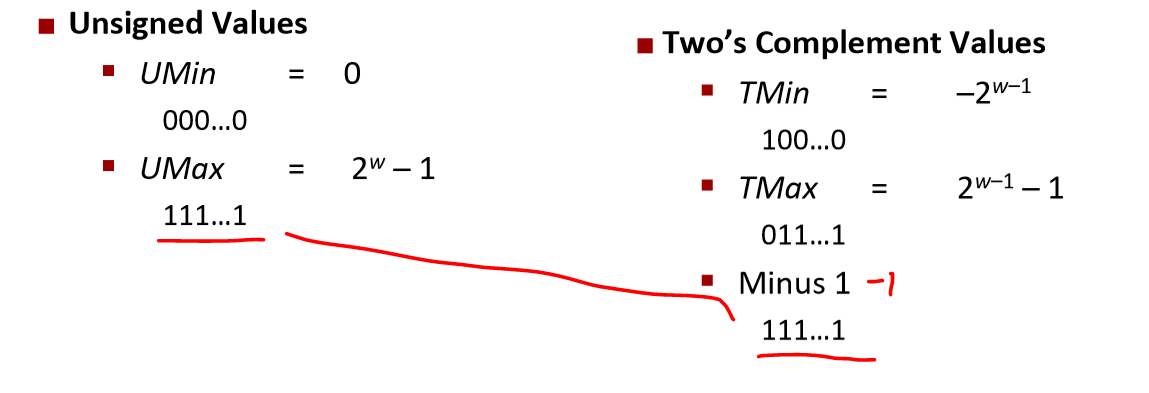
加法优先级比移位要高 1<<2+3<<4 = 1<<(2+3)<<4



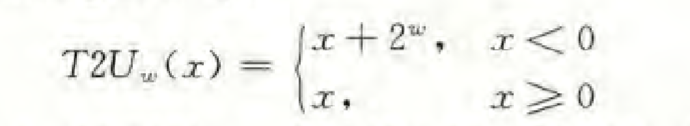
**2.2**

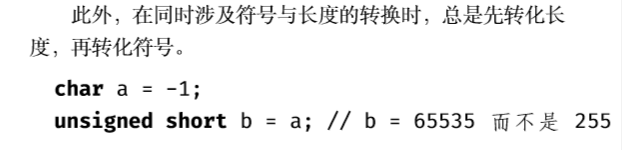
**·**





**·**有符号数和无符号数之间的转化：保持位模式不变，改变解释这些位的方式，值可能改变。



  
sizeof(int)是unsigned

C语言中，要创建一个无符号常量，必须加上后缀字符U或u。

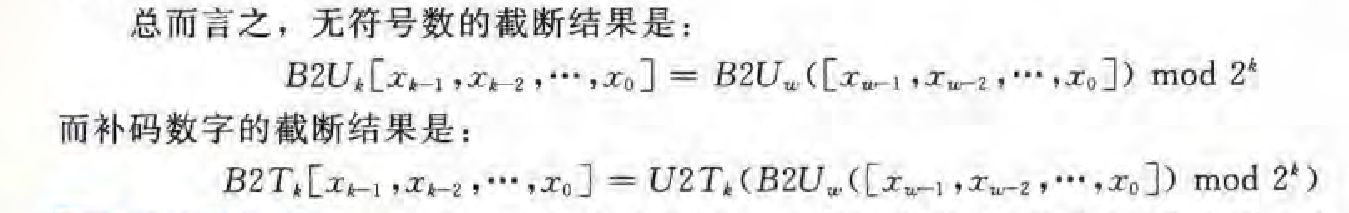
当执行一个运算时，如果它的一个运算数是有符号数而另一个是无符号数，那么C语言会隐式地将有符号数强制类型转化为无符号数。

**·**扩展一个数字：·无符号数——零扩展

·有符号数——符号扩展 值不变

**·**截断一个数字：·无符号数——mod 2^k

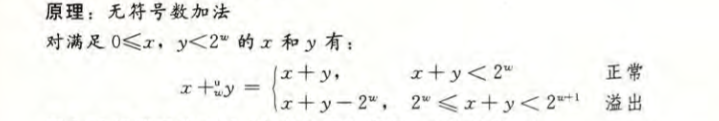
·有符号数——先转化为无符号数，再mod 2^k，再转化回有符号数



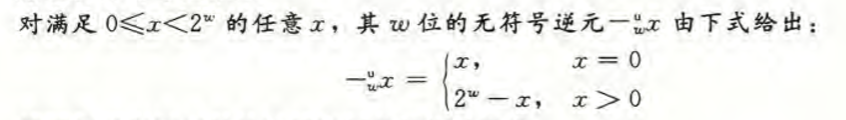
**2.3**

模数加法形成了一种数学结构，称为阿贝尔群。有一个单位元0，且每个元素都有一个加法逆元。

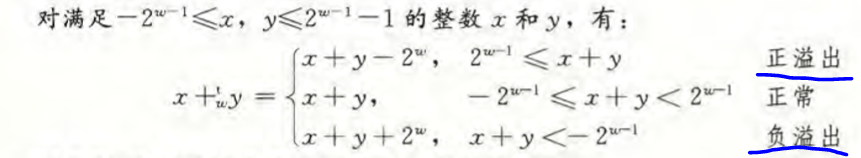
**·**无符号数加法：



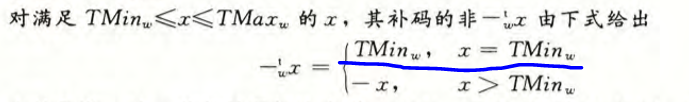
**·**无符号数求反：



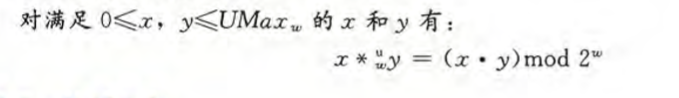
**·**补码加法：



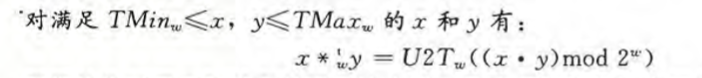
**·**补码求非：



**·**无符号数乘法：



**·**补码乘法：



无符号数和补码两种乘法乘积的全部位表示不同，但是截断后的乘积位级表示相同。

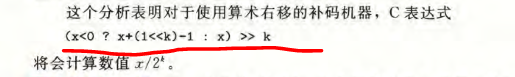
·通过移位代替乘法：u << k 等价于 u \* 2^k

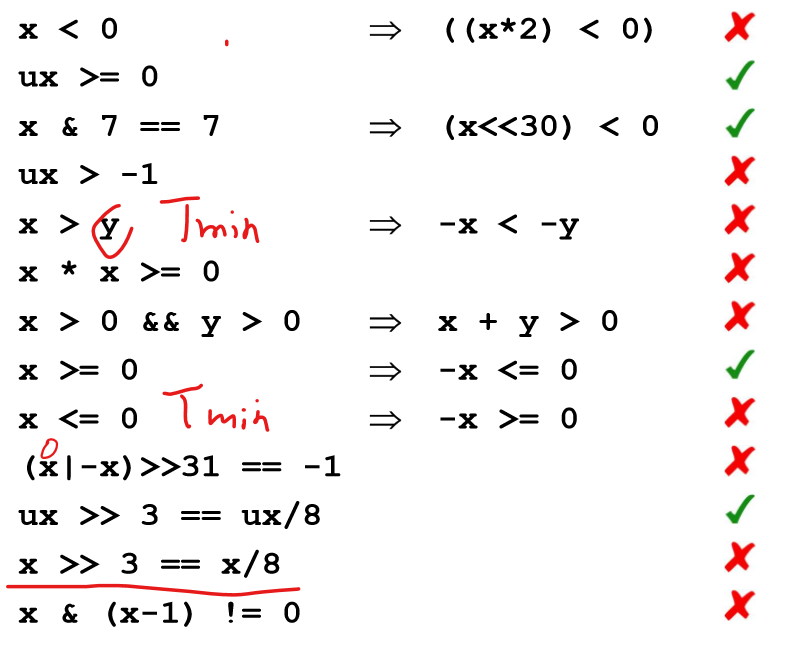
因为移位代价更小，用移位和加减法的组合代替乘法。

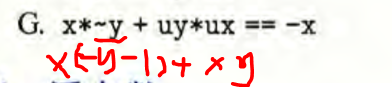
·通过移位代替除法：·无符号数：逻辑右移u >> k 等价于 u / 2k，向下/向零舍入

·补码：算数右移·x >> k 等价于x / 2k，向下舍入

·(x + (1<<k)-1) >> k 等价于 x / 2k，向上舍入





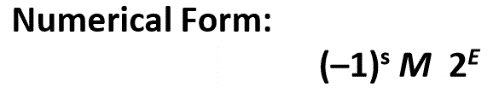


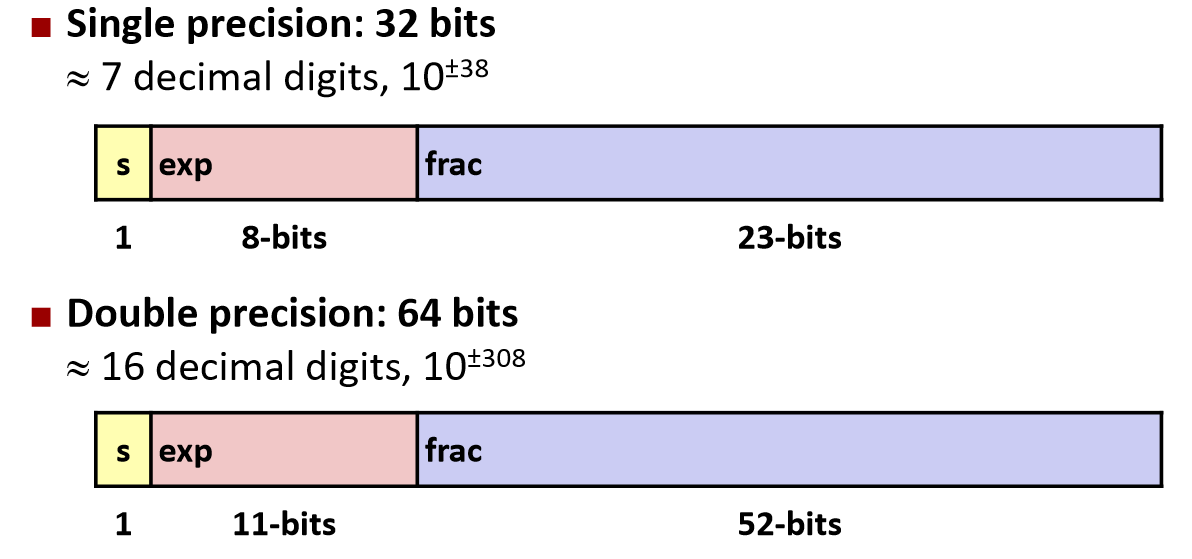
**2.4 浮点数**

**·**1.0 –ε代表接近1的数

**·**IEEE浮点表示：**·**符号（s）

**·**尾数M：一个二进制小数，范围是1~2-ε或0~1-ε

 **·**阶码E



根据exp的值，被编码的值可以分为三种不同的情况：

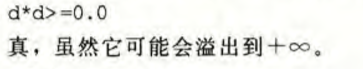
·规格化的值：exp的位模式既不全为0也不全为1。阶码的值E=e-Bias，e是无符号数（ek-1…e1e0），Bias是一个等于（2^(k-1）- 1）的偏置值（单精度是127，双精度是1023），k是exp的位数，尾数的值M=1+f。

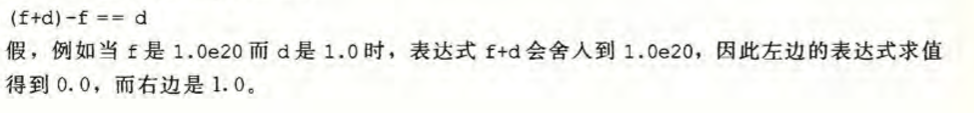
·非规格化的值：阶码域（exp）为全0，阶码的值E=1-Bias，尾数的值M=f。

用途：1、提供了一种表示数值0的方法。s=1或0，exp全0，f全0，有+0、-0两种，大部分情况下相同。

2、可表示那些非常接近0的数。f != 0。均匀分布

·特殊值：指阶码（exp）全为1。当小数域全为0时，得到的值表示无穷，s=0时为 +∞，s=1时为-∞，无穷表示溢出的结果。当小数域为非零时，结果为NaN，如根号-1，∞-∞，∞×0。





**第三章**

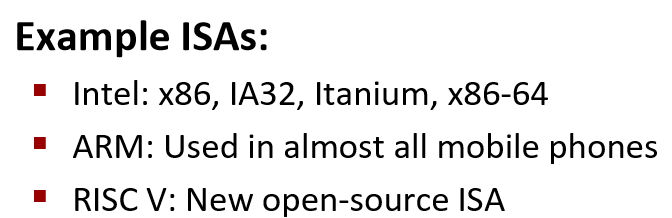
**3.1**

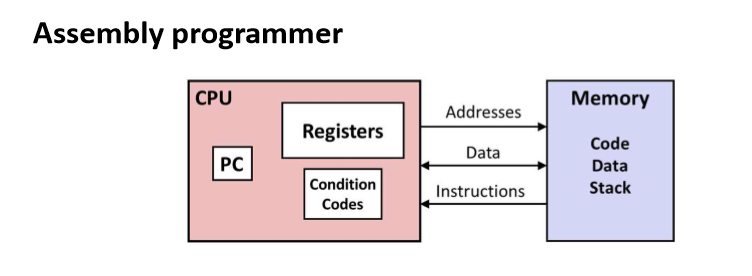
IA32 是 x86的前身

我们学的是x86-64

**3.2**

·指令集架构（ISA：Instruction Set Architecture）





·PC(Program Counter 程序计数器)：在x86-64中用%rip表示，给出将要执行的下一条指令在内存中的地址。

·整数寄存器文件（Register File）：包含16个命名的位置，分别存储64位的值

·条件码寄存器（Condition Code）：保存着最近执行的算术或逻辑指令的状态信息

·内存中包含：程序的可执行机器代码，操作系统需要的一些信息，运行时栈，用户分配的内存块。

Static libraries (.a)

·编译过程:

hello

hello.c

hello.i

hello.s

hello.o

链接器

Linker

汇编器

Assembler

编译器

Compiler

预处理器

Pre-Processor

可执行文件

源文件

目标文件

汇编文件

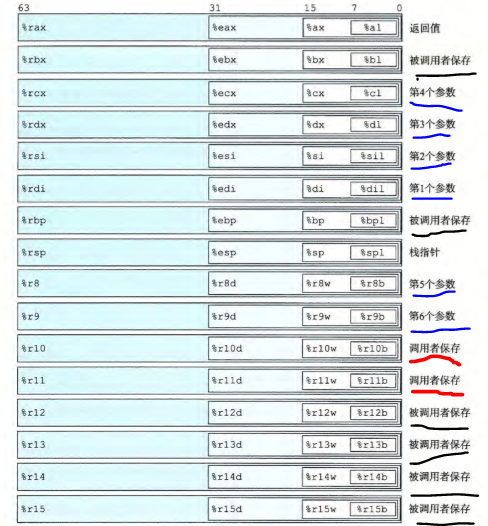
Modified源文件

·反汇编器：一类程序，可根据机器代码中的字节序列产生一种类似于汇编代码的格式。如objdump。不需要访问该程序的源代码和汇编代码。

**3.3**

X86-64体系架构中，8位为一个字节，16位表示字，32位表示双字，64位表示四字。





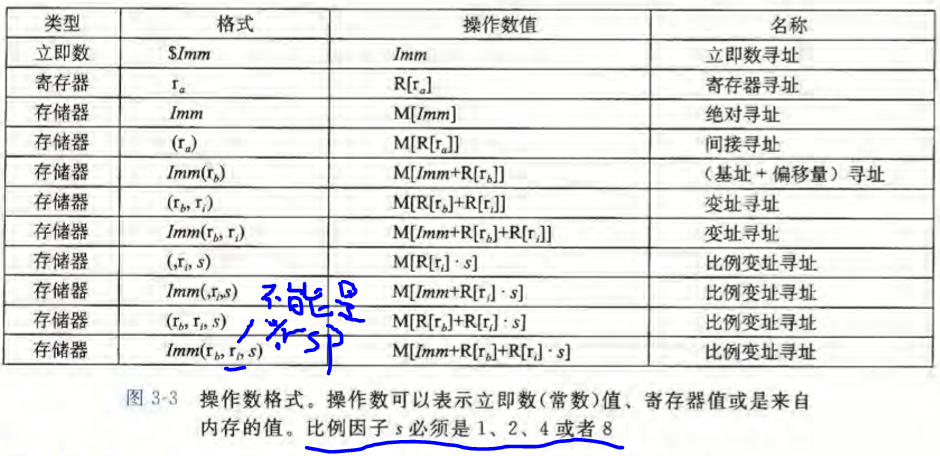
当以寄存器为目标时，对于生成小于8字节结果的指令，有以下两条规则：

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

**3.4**

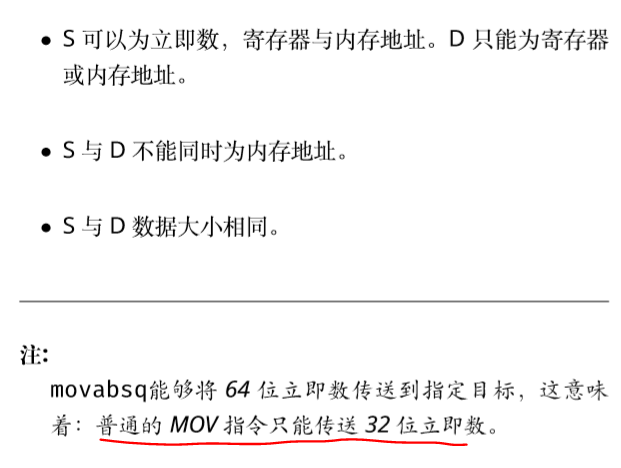
·操作数指示符

操作数分为三种：立即数、寄存器和内存引用，格式如下：



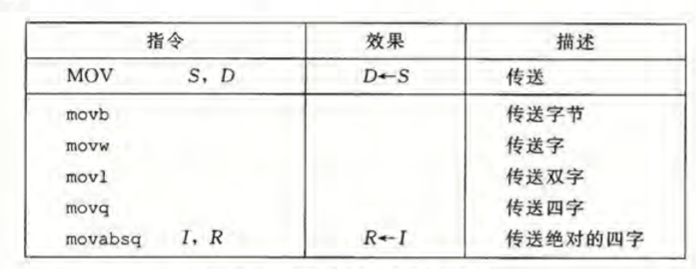
·数据传送指令

MOV类：源操作数和目的操作数指定的位置可以是寄存器或者内存位置，但不能都是内存位置。



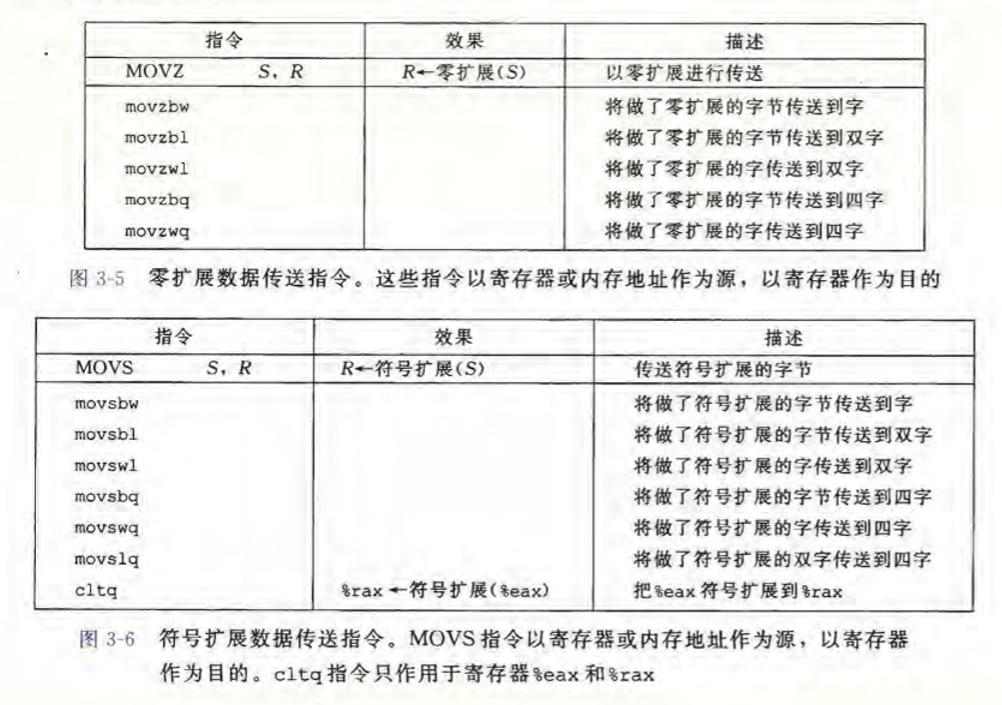
简单数据传送指令：

其中，movl指令以寄存器作为目的时，会把该寄存器的高位4字节设置为0。常规的movq指令只能以表示32位补码数字的立即数位源操作数并扩展到64位。而Movabsq指令能够以任意64位立即数值作为源操作数，且只能以寄存器作为目的。



两类数据移动指令：

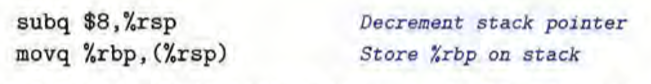
将较小的源值复制到较大的目的时使用。复制到目的寄存器。MOVZ类把目的中剩余的字节填充为0，而MOVS类通过符号扩展来填充。



·只有64位寄存器可以用作地址寄存器，即r开头的寄存器。

·压入和弹出栈指令：push和pop

pushq 等价于



但pushq指令编码仅一个字节而后者占8个字节。

**3.5算数和逻辑操作**

·加载有效地址（load effective address）leaq

leaq S D：将S的有效地址写入D。D必须为一个寄存器

movq S D：将S中的值写入D。

巧妙用法：简单算术表达式。

·一元操作 D：寄存器或内存地址

|  |  |  |
| --- | --- | --- |
| INC D | D+1 -> D | 加1 |
| DEC D | D-1 -> D | 减1 |
| NEG D | -D -> D | 取负（取反+1） |
| NOT D | ~D -> D | 取补（取反） |

·二元操作 D：寄存器或内存地址 S：寄存器、操作数或内存地址

|  |  |  |
| --- | --- | --- |
| ADD S,D | D+S -> D | 加 |
| SUB S,D | D-S -> D | 减 |
| IMUL S,D | D\*S -> D | 乘 |
| XOR S,D | D^S -> D | 异或 |
| OR S,D | D|S -> D | 或 |
| AND S,D | D&S -> D | 与 |

·移位操作

先给出移位量，然后第二项给出要移位的数。移位量可以是一个立即数，或者放在单字节寄存器%cl中。当移位量放在%cl中时， salb，salw，sall，salq的移位数分别为%cl中存的数的低3位、低4位、低5位、低6位所代表的十进制数。

|  |  |  |
| --- | --- | --- |
| SAL/AHL K,D | D<<K -> D | 左移 |
| SAR K,D | D >>A K -> D | 算术右移 |
| SHR K,D | D >>L K -> D | 逻辑右移 |

·八字（oct word）

**3.6**

·条件码

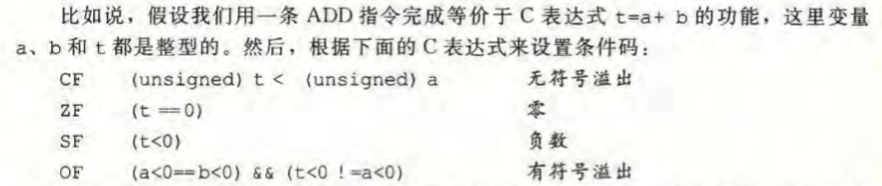
CF：进位标志。可用来检验无符号操作的溢出

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

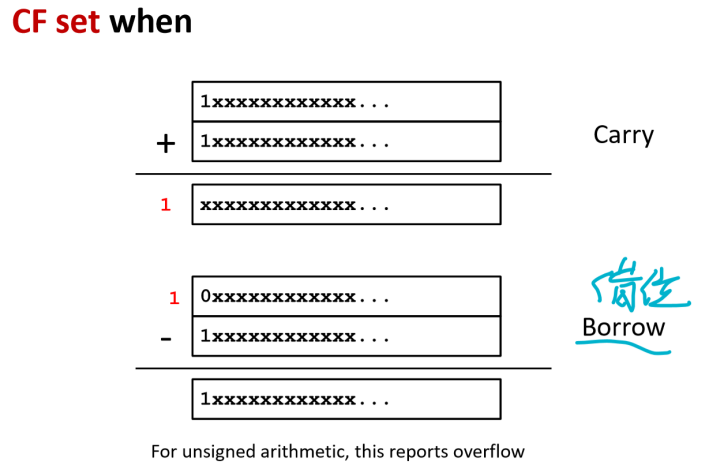
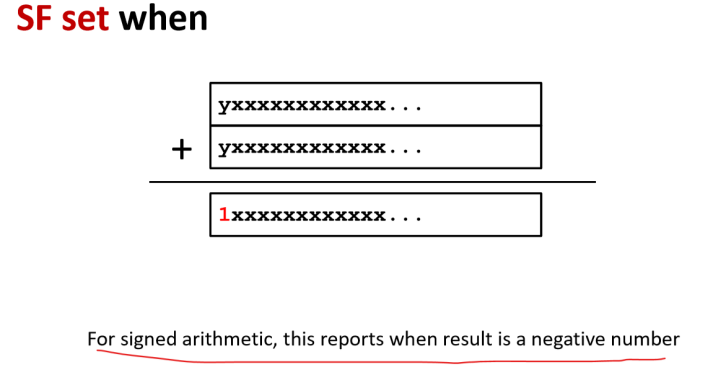
OF：溢出标志。最近的操作导致补码溢出。（正溢出/负溢出）

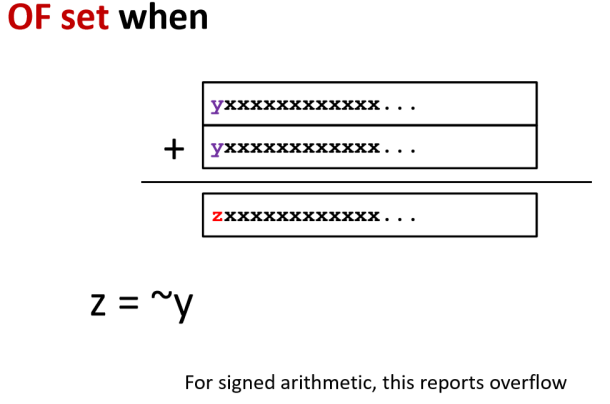
除leaq指令外所有指令都会设置条件码。对于逻辑操作，CF和OF为0。对于移位操作,OF为0，CF为最后一个被移出的位。



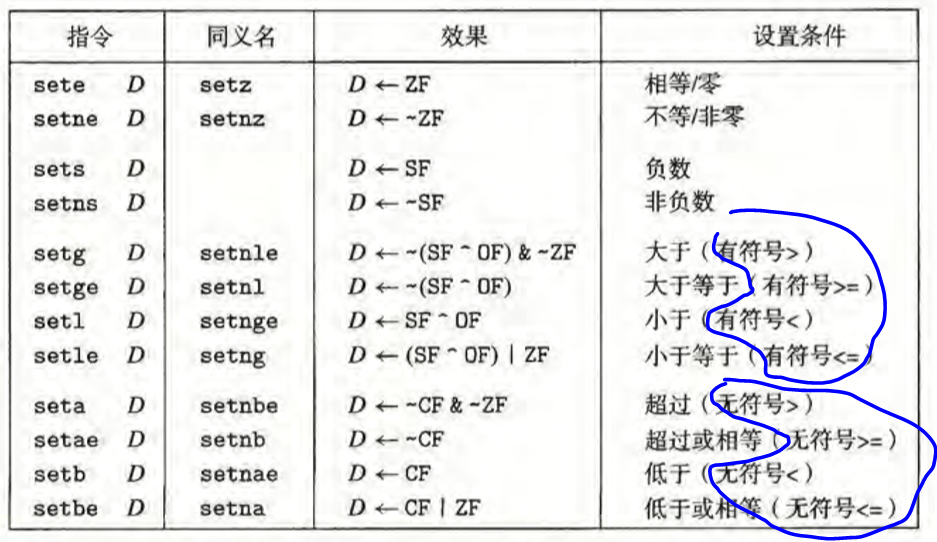
两种只改变条件码而不改变任何其他寄存器的指令：CMP 和 TEST。

CMP与SUB一样，TEST与AND一样

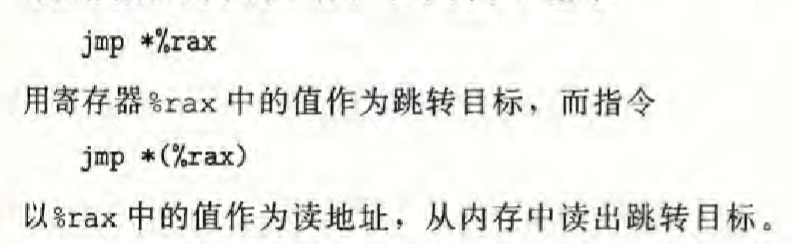
 

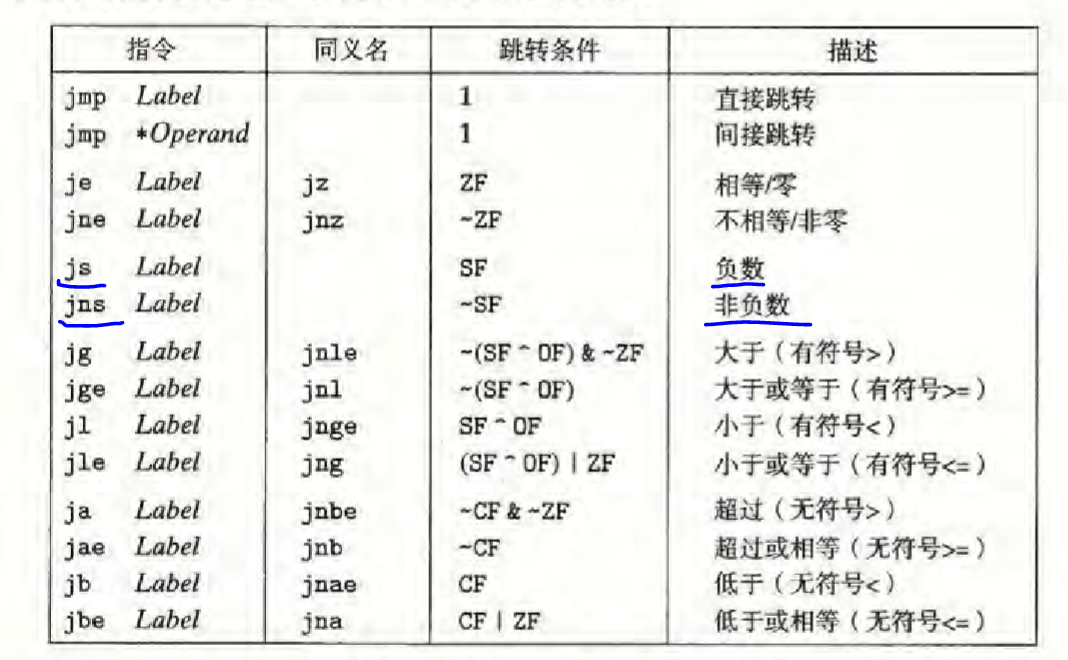


·SET指令：目的操作数是低位单字节寄存器元素，或一个字节的内存位置。



·跳转指令





跳转指令都是需要两个字节。

·用条件传送来实现条件分支，基于条件传送的代码会比基于条件控制转移的代码性能好。

不是所有的条件表达式都可以用条件传送来编译。使用条件传送也不总是会提高代码的效率。

·编译代码不仅仅是C代码的直接翻译，还可能优化

·gcc将while循环翻译为机器代码的两种方法：

1. 跳转到中间
2. Guarded-do