# 深入理解计算机系统

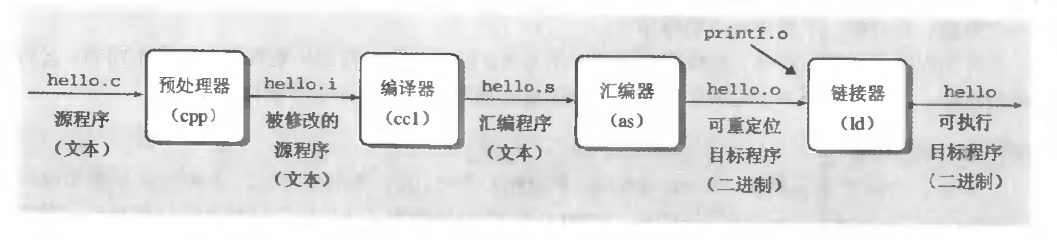
# 第一章：系统漫游

### 程序翻译

Unix > gcc -o hello hello.c

gcc编译器驱动程序读错源文件 hello.c 翻译为可执行目标文件hello

该翻译过程一共分为4阶段。执行这4个阶段的程序（预处理器(cpp)，编译器(ccl)，汇编器(as)，连接器(ld)）一起构成了编译系统。



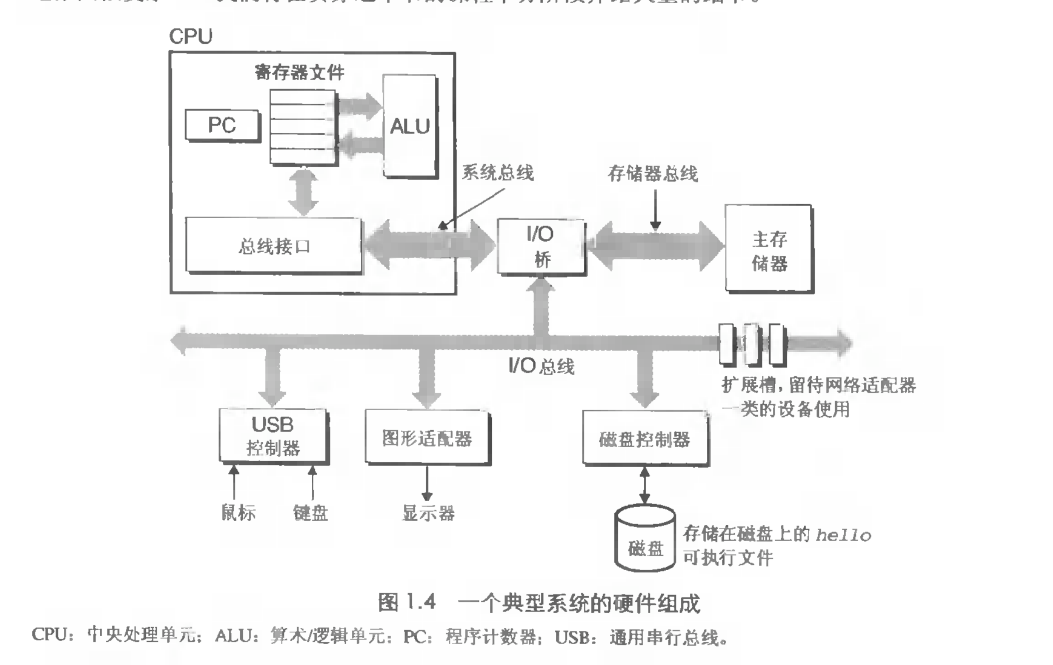
**预处理阶段:**根据字符#开头的命令。修改原始的C文件。如#include<stdio.h>指令会告诉预处理器读取系统头文件stdio.h的内容，并将它直接插入到程序文本中。得到另外一个C文件，以.i作为文件扩展名

**编译阶段：**编译器将文本文件.i翻译为.s的汇编语言程序。 汇编语言程序每条语句都以标准的文本格式描述一条低级机器语言指令。 汇编语言为不同的高级语言的不同编译器提供了通用的输出语言。

**汇编阶段：**汇编器将.s文件翻译为机器语言指令，把这些指令打包成为一种**可重定位（relocatable）**目标程序的格式并保存在.o文件中。该文件的字节编码是机器语言指令而不是字符。

**连接阶段：**我们的hello程序调用了printf函数，该函数存在于名为printf.o的编译目标文件中，而这个文件必须以某种方式并入到hello.o的程序中。连接器负责并入。最终生成一个可执行文件

### 系统的硬件组成



**总线：**贯穿整个系统的是一组电子管道，称为总线,负责携带信息字节并在各个部件之间传递。通常总线传送定长的字节块，也就是字。不同的系统，字长不同。32位系统字长为4个字节，64位系统字长为8个字节。

**I/O设备：**I/O设备是系统与外界通信的通道。如键盘，鼠标，显示器。每个IO设备都是通过一个控制器或适配器与IO总线相连。控制器是系统电路板上的芯片组，适配器是一块插在主板插槽上的卡。

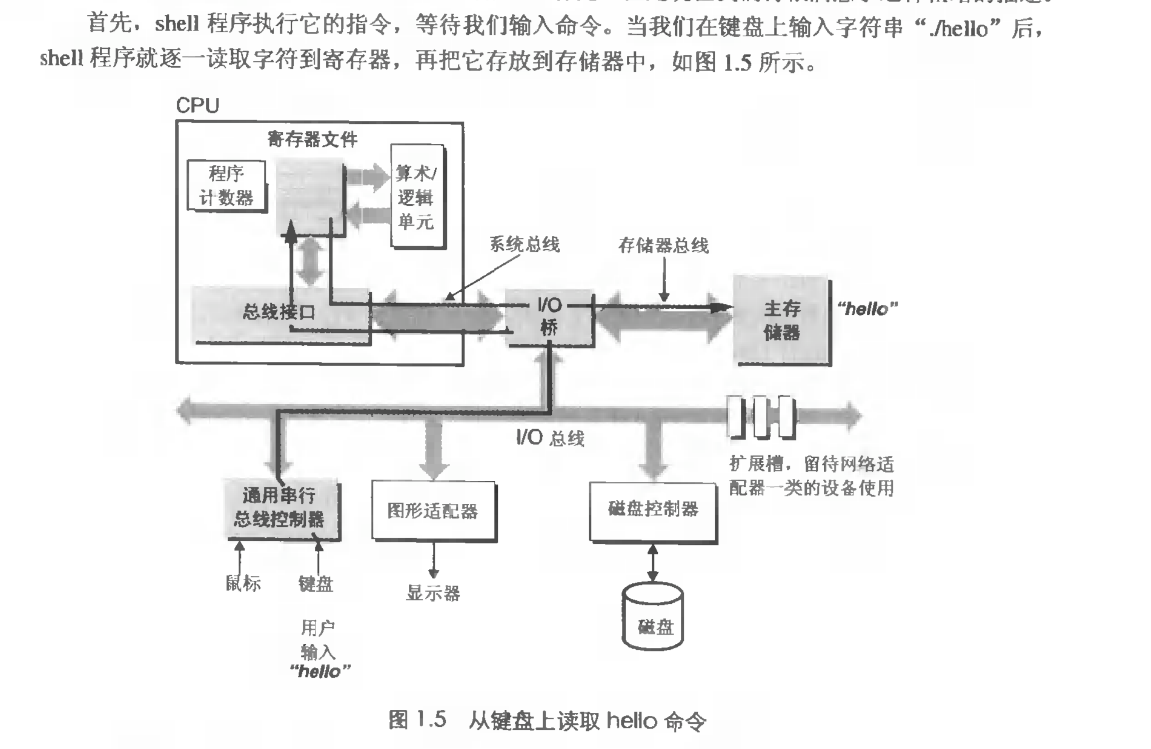
**主存：**主存是一个临时储存设备，在处理执行程序时，用于存放程序和程序处理的数据。物理上主存是有一组DRAM（动态随机存取存储器）芯片组成。逻辑上，存储器是由一个线性的字节数组组成，每个字节都有自己唯一的地址（或索引），这些地址是从0开始。

**处理器：**CPU简称处理器，是执行存储在主存中指令的引擎。 处理器的核心为被称为程序计数器（PC）的字长大小的寄存器。任何时候，PC都指向主存中的某条机器语言指令（即PC存储的是主存中的地址）。

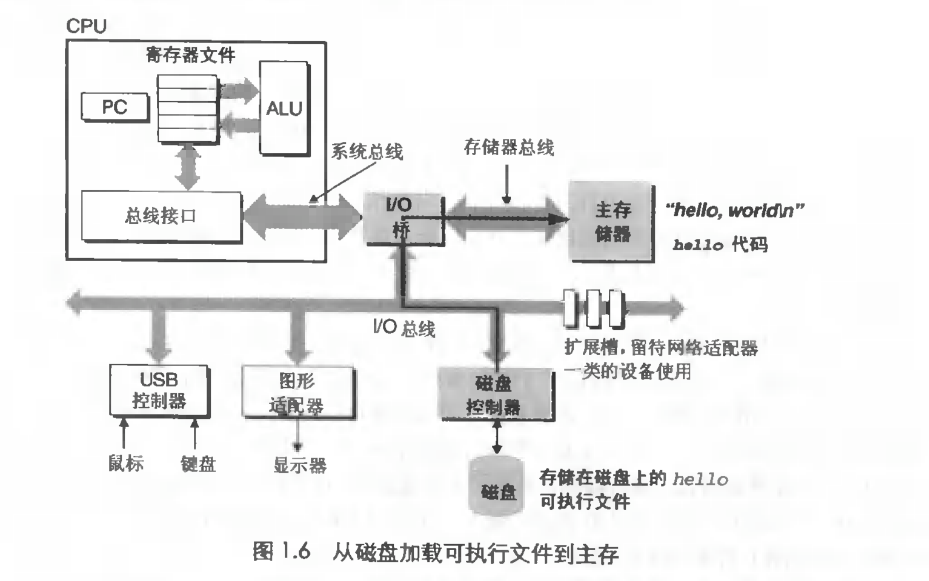
CPU执行顺序： 从CP中指向的存储器读取指令，解释指令中的位，执行指令，然后更新CP的下一条指令，改指令不一定与刚执行的战令相邻。

寄存器文件是一个小的存储设备，由一些字长大小的寄存器组成，每个寄存器拥有唯一的名字。ALU用于及计算新的数据和地址值。

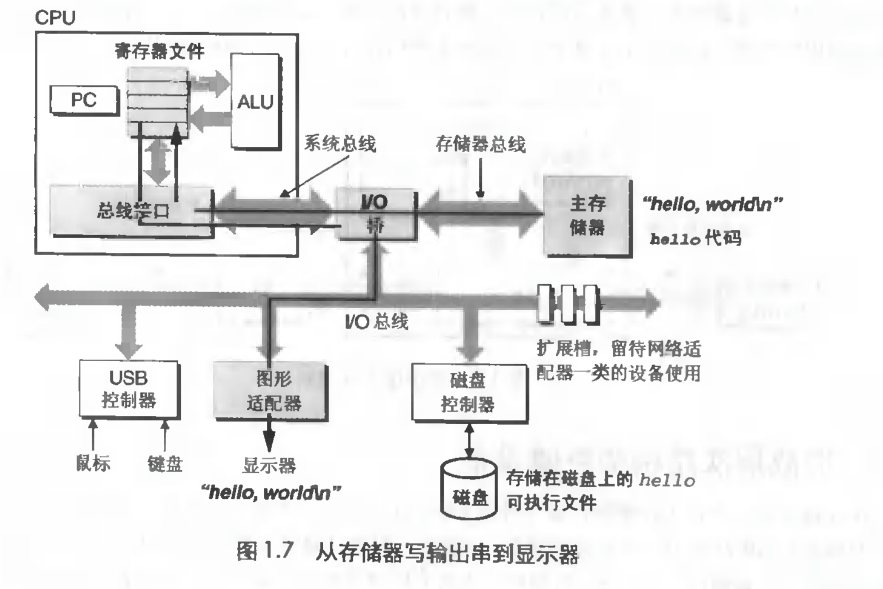
### 执行Hello程序



当我们在键盘上敲回车时，shell知道我们命令输入结束。然后shell执行一系列指令，这些指令将hello目标文件中的代码和数据从磁盘拷贝到主存，从而加载hello文件。



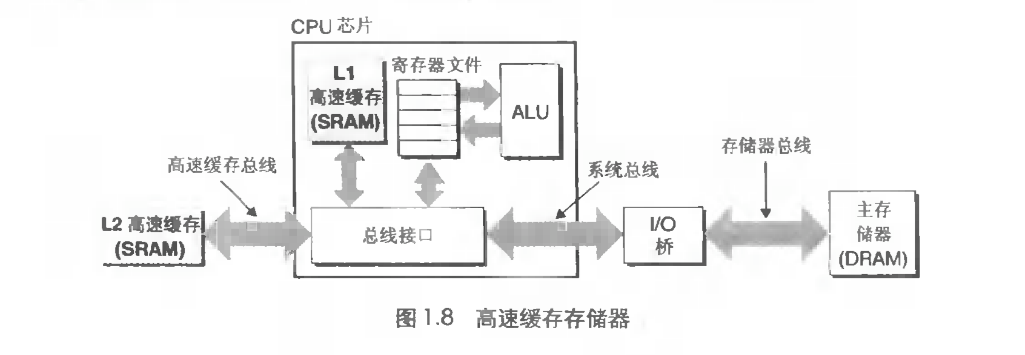
当hello目标文件的代码和数据加载到了存储器，处理器就开始执行hello程序中的机器语言指令。printf指令将“hello world\n”的字节从存储器拷贝到寄存器，再从寄存器文件拷贝显示设备，最终显示到屏幕上。

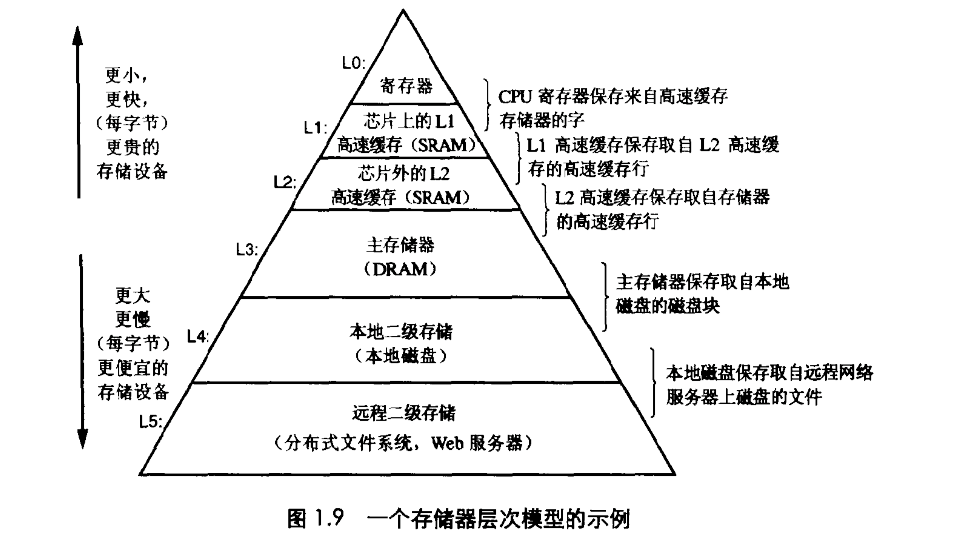


### 高速缓存

hello程序的指令，最初是存在磁盘上，当程序被加载时它们被拷贝到主存。当处理器运行时，它们又被拷贝到主存中。大量的拷贝操作减慢了程序的实际工作。 高速缓存的目的是为了使得这些拷贝操作尽可能的快。

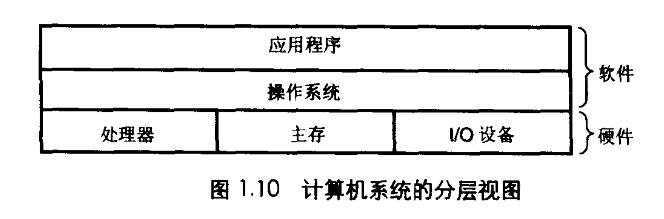
CPU从寄存器读取速度比从主存中读取快100倍。针对这种处理器与主存之间的差异，系统设计者使用了更小更快的设备，称为**高速缓存存储器**，用来作为暂时的存储器。L1高速缓存的容量，可容纳数万字节，访问速度几乎和寄存器文件一样。L2高速缓冲器是通过一条特殊总线连接到CPU，访问速度为L1的5倍但仍比访问主存快5-10倍。





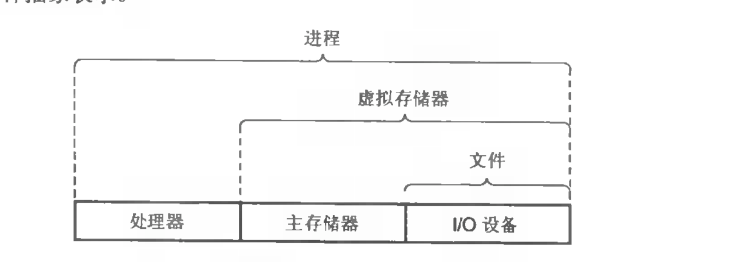
### 操作系统管理硬件

程序依靠操作系统的服务访问。操作系统是应用程序和硬件之间的一层软件，软件对硬件的操作必须通过操作系统。



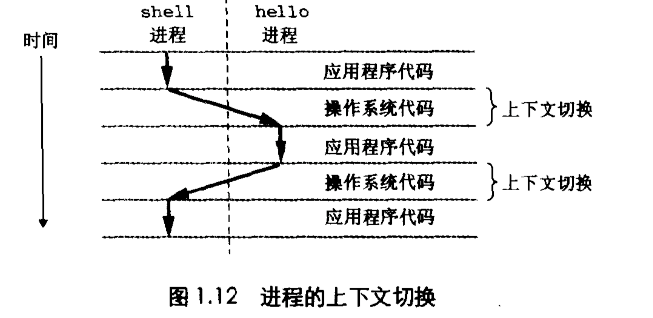
操作系统有两个基本功能：1.防止硬件被软件滥用 2.在不同的低级设备中为程序提供统一的方法。

操作系统通过几个抽象概率（文件，虚拟存储器，进程）实现这两个功能。



**进程：**进程是操作系统对运行程序的一种抽象。一个系统可以同时运行多个进程，而每个进程都好像单独地占用硬件。我们称之为并发运行，但实际上一个进程的指令和另一个进程的指令是交错执行。操作系统实现交错执行的机制称为**上下文切换**。

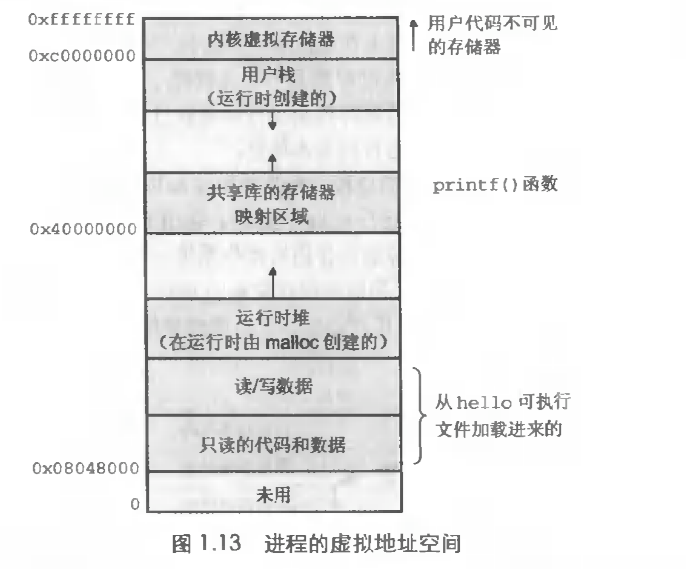
操作系统会保存进程运行所需的所有状态信息，包括PC和寄存器当前值以及主存的内容，这种状态也就是**上下文**。在任何时刻，系统都只有一个进程在运行。上下文切换时，操作系统会保存当前上下文，恢复新进程的上下文，然后将控制权交给新进程，新进程从它停止的地方开始。



**线程：**在现代系统中，一个进程可以由多个称为线程的执行单位组成。每个线程都运行在进程的上下文中，并共享相同的代码和全局数据。线程成为越来越重要的编程模式，因为多线程比多进程之间更容易共享数据，执行速度也更快。

**虚拟存储器：**虚拟存储器是一个抽象概念，为每个进程提供一个假象，好像每个进程都在独占使用主存，每个进程看到的存储器都是一致的。基本思想是将进程虚拟存储的内容存储在磁盘上，主存作为磁盘的高速缓存。

下图为Linux进程的虚拟地址空间，注意最下面的地址为0



从下往上看

·程序代码和数据：代码是从同一固定地址开始，紧接着是C全局变量对应的数据区。代码和数据区是有可执行目标文件直接初始化得到。

·堆：代码和数据区挨着的是运行时堆。代码和数据在一开始运行就被指定了大小。像malloc和free这种C标准库函数的结果，堆可以动态的扩展和收缩。

·共享库：地址空间中间部分，用来存放标准库和数学库这样共享库的代码和数据。

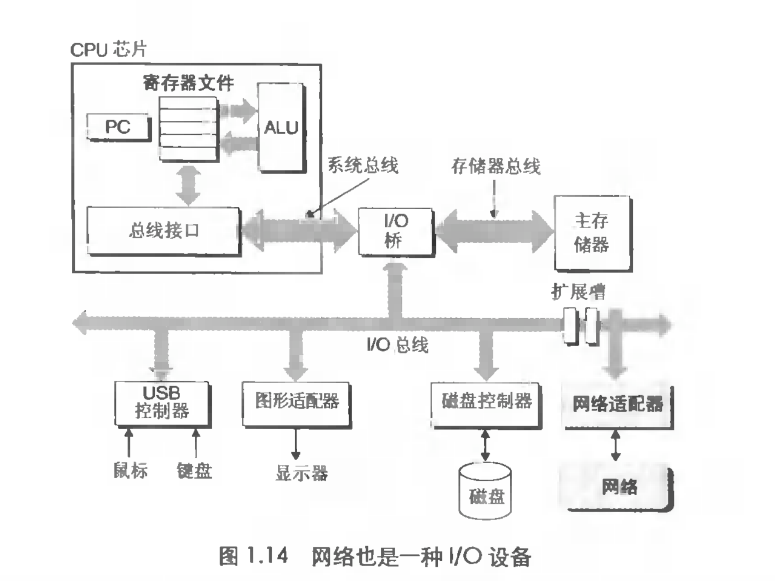
·栈：位于虚拟空间顶部，编译器用它实现函数调用。和堆一样，栈可以在运行时动态扩展和收缩，调用函数时，栈会增长，返回时，栈会收缩。

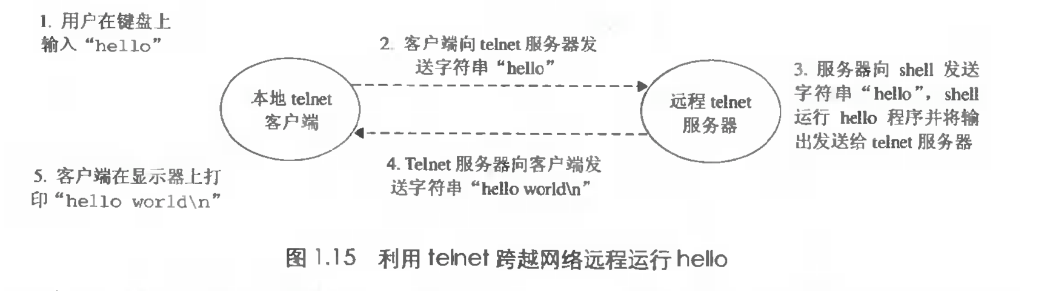
·内核虚拟存储器：内核是操作系统操作系统驻留在存储器中部分。顶部1/4位内核预留。应用程序不允许读写该区域。

**文件：**文件就是字节序列。每个IO设备，甚至网络都可以被当成文件。系统中所有的输入输出都是通过读写文件完成的。

### 网络系统和其他系统通信

从一个单独的系统来看，网络可以视为一个IO设备





**总结**

1. **计算机系统是有硬件和系统软件组成，共同协作以运行应用程序。**
2. **程序被翻译为不同形式，最开始为ASCII文本，然后被汇编器和连接器翻译为二进制可执行文件**
3. **操作系统是应用程序和硬件之间的媒介。**
4. **操作系统的三个基本的抽象概念：**
   1. **文件：对IO设备的抽象**
   2. **虚拟存储器：对磁盘(IO设备)和主存的抽象**
   3. **进程：对CPU，主存和IO设备的抽象**
5. **网络也是一种IO设备**

# 第二章：程序执行与结构

### 信息存储

大多数计算机以**字节**作为最小可寻址的存储器单位（地址还是32位，但是存储的内容位字节（8位）大小）。

指针：值是指向某个对象的位置，类型表示那个位置上所存储对象的类型。

16进制转2进制：将16进制的每个值，转化为4位2进制

2进制转16进制：每4个二进制转化为一个16进制。

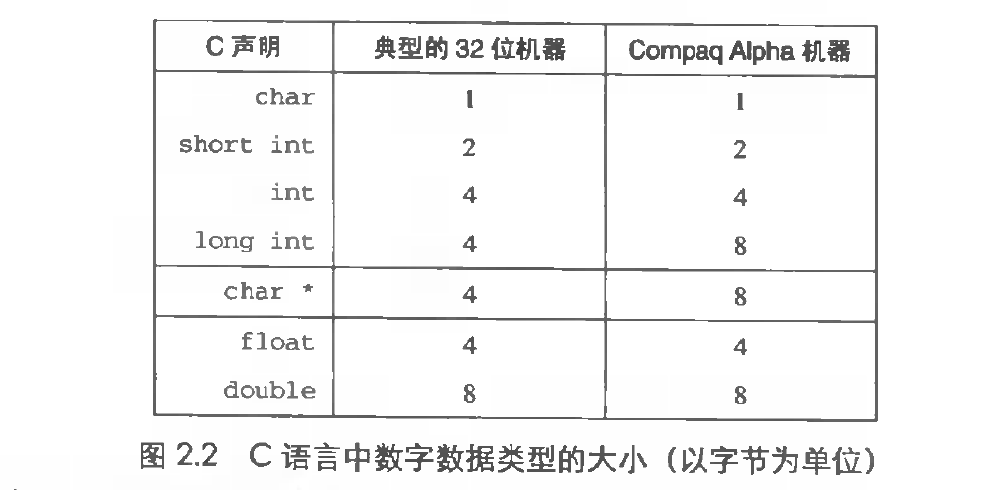
当x为2的n次幂时，表示为2进制则是 1后面带n个0

10进制转16进制：反复除以16得到商q和余数r，然后r作为最低为，q继续除

16进制转10进制：相应的16的幂乘以每个对应的16进制的数

字长：指针数据的标称大小。一个字长为n的机器，虚拟地址空间为0-2^n,32位机子字长位4个字节。64位机子字长位8个字节。

32位和64位不同类型的大小。（注意：long int 和 char\* 大小不同。 Char\* 指针大小与机器字长大小相等）



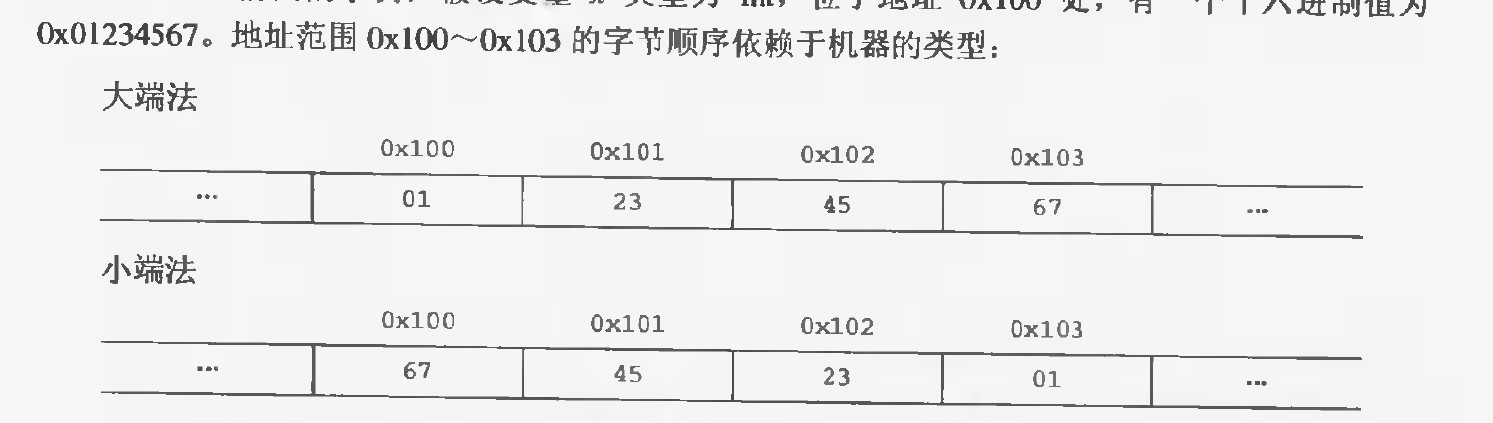
### 寻址和字节顺序

寻址：对象的地址为所使用的字节序列的最小地址。如一个int类型的x地址为0x100；那么它的数据存储在0x100，0x101,0x102,0x103

字节顺序：

小端法：最低有效位在最前面。（Intel的机器）

大端法：最高有效位在最前面。（IBM，SUN）



对于大多数程序来说，字节顺序是无感知的。 但在网络信息传输时，小端机器和大端机器之间通信将会造成影响。所以网络程序代码必须遵循已建立的字节顺序规则。

**字符串表示：**终止字节的为0x00 字符串表示没有大小端

“123456” = 31 32 33 34 35 36 00 因为字节顺序无关，所以文本比二进制数据具有更强的平台独立性

**交换数值：** 通过a^a = 0 实现不需要新变量实现交换 a^0x00= a a^0xff = ~a

a = a^b;

b = a^b;

a = a^b;

**左移：**x<<k 会生成一个值，x向左移动k位，丢弃k个最高位，并在右端补k个0 注意优先级 x<<5-1 等于 x<<(5-1)

**右移：**x>>k 右移分为 逻辑右移和算术右移。 逻辑右移则在最高位补0，用于无符号计算。 算术右移则会在左边补k个最高有效位拷贝，这对有符号整数数据运算中非常有用。

**补码：**正数的补码是正数本身，负数的补码是负数对应的二进制取反再+1 如：0x80 补码为 0x00 +1 =1 结果为-1。 要+1的原因是0xff 为负数最小位，单独取反的话值为0 与0x00 取值相同，所以+1以做区分

负数表示：-x = ~(x-1) + (2^n)

**负数和无符号数之间转换：**对于0<=x<2(w-1)之间的数，转换不变，对于其他的转换时 加上或减去 2(w). eg： -1 表示为 ff ff 表示为无符号为：2w-1 = 65535 ffff无符号转换为有符号为： 65535 - 65536 = -1

对于有符号和无符号的运算，C会默认当成无符号的数进行计算，计算时无差异，但比较时可能会出错：

Eg：-1 < 0U false

2147483647U > -2147483647-1 false

2147483647 > (int) 2147483648U true

**扩展：**扩展一个数，会进行符号扩展，在表示中添加最高有效位的值。

**截断：**截断一个数，会直接丢弃掉超过的位数。

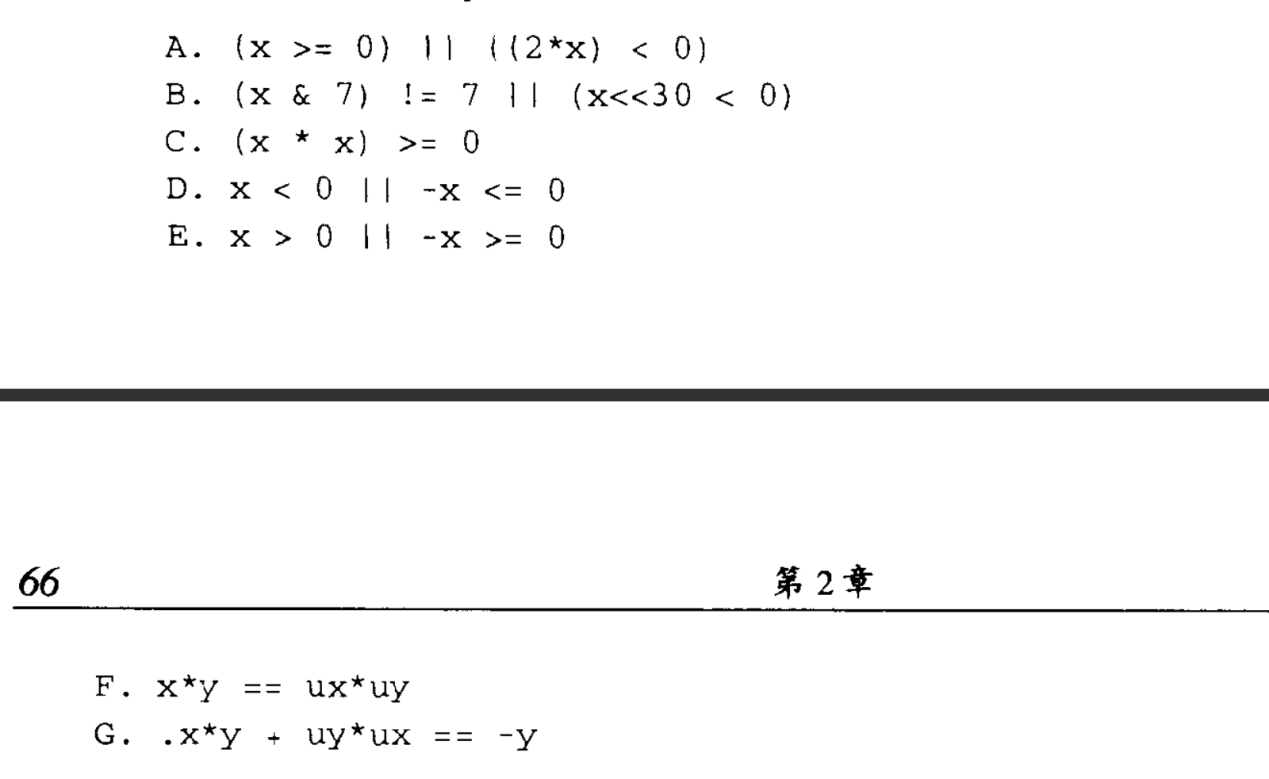
**除法和右移：**对于正数来说除法和右移都是向0靠近，如7/2 = 7>>1 = 3

对于负数来说则不对 -7/2 = -3 向0靠近

右移 -7 = 0x1001 -7>>1 =0x1100 = -4 向下取整

除法都为向0靠近，右移都为向下取整

可以使用偏移让右移与除法一致： x+(1<<k -1) >>k



A为false：乘法为截取，x = 0x80000000 时为false

B为true: x值为 0x......7 向右移30位 符号位必为1 结果为负

C为false: x值大时将会溢出，截取32位后不一定为正数

D为true

E为false：0x80000000 的-x为0x80000000

F为true: 无论有符号乘法还是无符号乘法，结果位数是一样的，两者比较时按无符号比较，结果一致。

## 浮点数

**计算公式：**

S:符号位，1为负，0位正

E:指数位，float中为8位，double中为11位。

M:有效位，float中为23位，double中为52位。

**格式化值：**

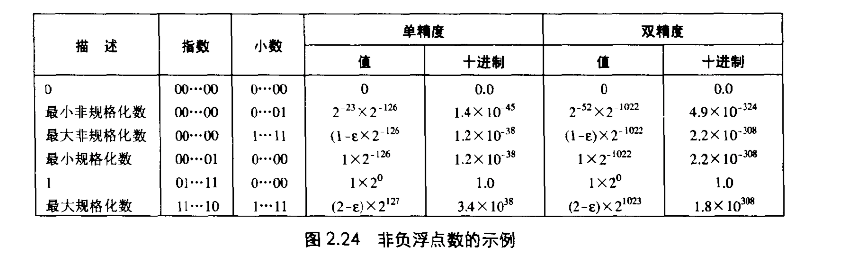
当E不是全0且不是全1时，就属于格式化值。此时E=e-Bias e为无符号数，二进制表示的值， Bias为偏移量，2^（k-1）-1 对于float Bias的值为 128-1 = 127 E的范围为 -126~127。 M = 1+f f为小数位的值 隐含以1开头，可以节省1位 M值为 1<=M<2

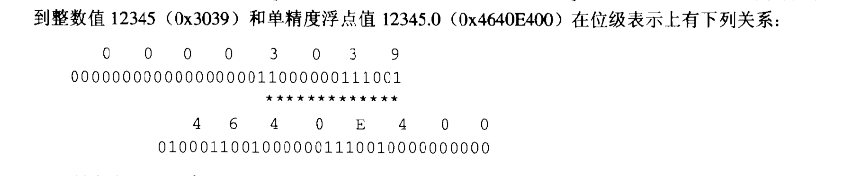
**非格式化值：**

当E全为0时： E = 1-Bias M = f

PS:为什么E不为-Bias？ 因为E为-Bias时，M为隐含带1的值，这样就不能表示小于 2^（-Bias）的值，也不能表示0了。 指数和有效位全为0时，符号为0/1 表+0 和 -0 在某些方面认为是不同的（1/0 为正无穷，1/-0 为负无穷），在其他方面是相同的。

当E为全1时：小数域全为0时表示正无穷





整数转浮点数：符号位不变，指数值变为位数+Bias，小数部分为整数去掉最高位后的值

舍入方式：向0取整，向下取整，向上取整，向偶数取整。

向偶数取整有个好处，可以在计算平均值时避免统计偏差。 因为向上会导致结果偏大，向下会导致结果偏小。

## 第三章：程序的机器级表示

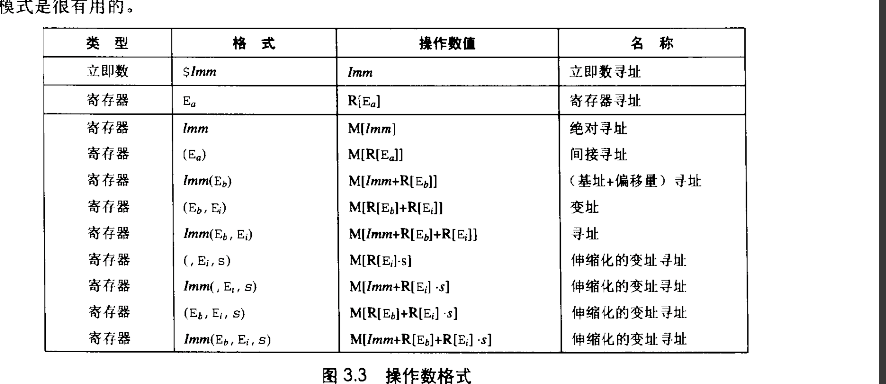
8个寄存器：每个寄存器都是32位 前3个寄存器（%eax,%ecx,%edx）和后三个寄存器（%ebx,%edi,%esi）的保存和恢复惯例不同，最后两个寄存器（%ebp,%esp）保存着指向程序栈中重要位置的指针，只有根据标准规则才可以修改这两个寄存器中的值。

操作数：

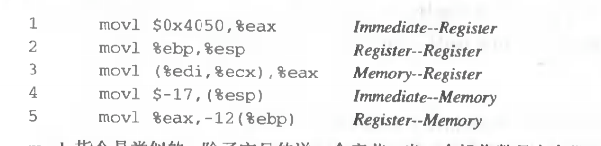
立即数：常数值，用$表示。 $0X1f

寄存器：表示某个寄存器中的内容，用%表示， %eax

存储器引用：根据计算出来的地址访问存储器位置。



MOV指令五种可能组合



压栈和出栈操作：栈因为是从高地址往低地址扩展，所以压栈顶部地址-4 出栈顶部地址+4

