1. **编码**
   1. **ASCII码**

用0-127来表示所有字符，但是只有美帝能用，其他国家语言放不进去，所以就需要扩充的字符编码集

* 1. **ANSI码**

是对ASCII的扩展，0-127与ASCII相同，128之后是某个国家语言的所有字符，ANSI编码是很多国家编码的总称：中国制定了GB2312编码，用来把中文编进去，如本的Shift\_JIS等等，各个国家有各自的标准，不同语言之间的ANSI码不能互相转换，这就会导致多语言混合的文本中会有乱码

* 1. **Unicode**

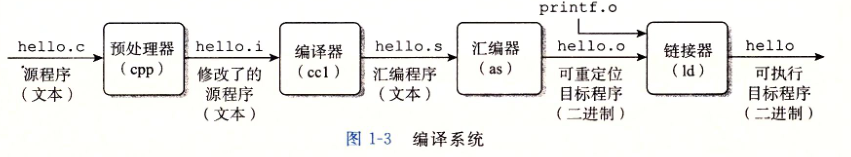
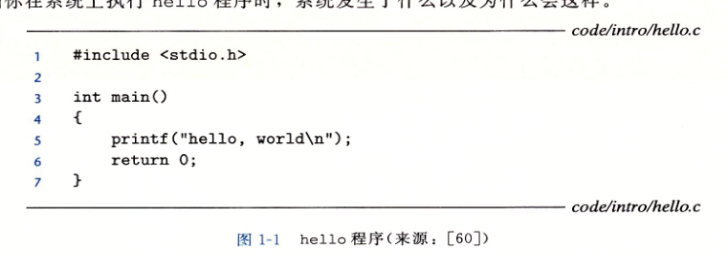
为了解决不同国家ANSI码冲突的问题，全世界每一个符号都给予一个独一无二的编码，最常用的是用两个字节表示一个字符（如果非常偏僻的需要4个字节）。但是问题在于，原本可以用一个字节存储的英文字母在Unicode中必须存储为两个字节，会造成浪费。

* 1. **UTF-8**

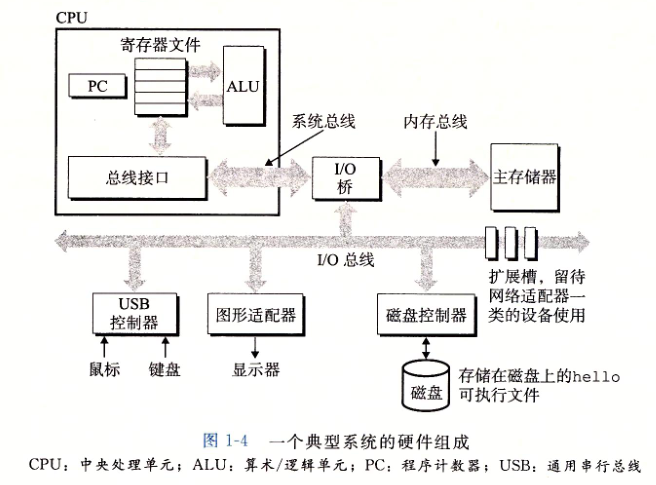
这是一种变长的编码方式，可以使用1-4个字节表示一个符号，当字符在ASCII范围时，就用一个字节表示，注意一点，Unicode中文占两个字节，UTF-8占3个字节，从unicode到utf-8不是直接对应，而是要通过一些算法和规则来转换

* 1. 在计算机内存中，统一使用unicode，当需要保存到硬盘或者传输时，转为UTF-8，用记事本编辑时，从文件读取的UTF-8被转为Unicode，编辑完成后，保存时再转为UTF-8.

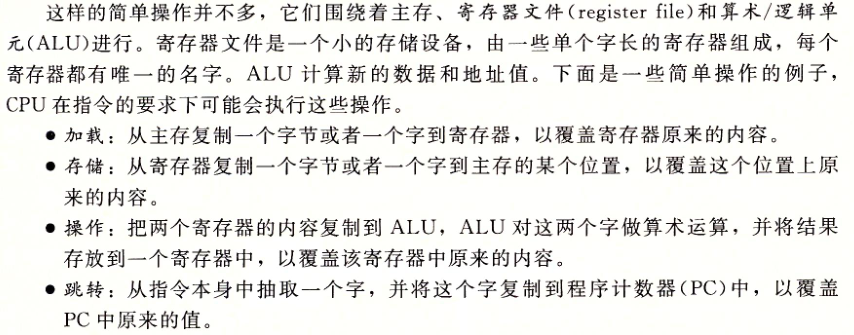
1. **源文件到可执行程序过程：**
   1. 预处理阶段：通过预处理器，将#的文件插入文本中，生成hello.i
   2. 编译阶段：通过编译器，转为汇编程序，hello.s
   3. 汇编阶段：汇编器将hello.s翻译成机器语言指令，打包成一种叫做可重定位目标程序的格式，hello.o
   4. 链接阶段：由于hello程序调用了printf函数，他是每个C编译器都提供的标准C库中的一个函数。Printf函数存在于printf.o的单独目标文件中，连接器就负责处理这种合并，结果得到hello文件，它是一个可执行目标文件，可以被加载到内存中，由系统执行.

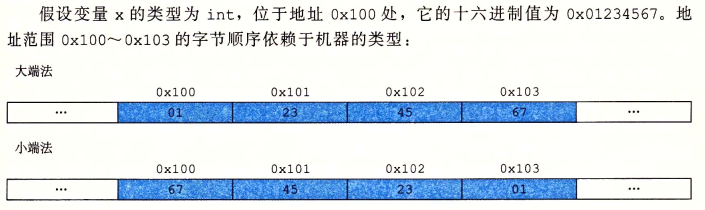


1. **系统的硬件组成**



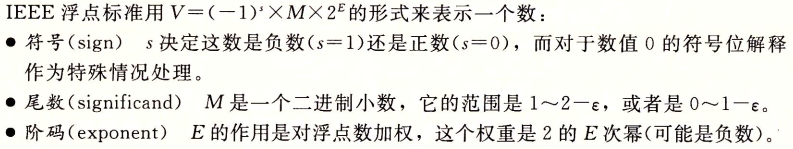
* 1. **总线**：贯穿整个系统的一组电子管道，它携带信息字节并负责在各个部件间传递，通常被设计成传送定长的字节快，也就是字（word），字中的字节数（即字长）要么是4个字节（32位）要么是8个字节（64位）
  2. **I/O设备**：键盘、鼠标、显示器和磁盘。通过控制器或适配器与I/O总线相连，控制器是I/O设备本身或系统的主印制电路板（主板）上的芯片组，适配器是一块插在主板插槽上的卡，左右都是总线和设备之间传递信息.
  3. **主存**：即内存，从逻辑上来说是一个线性的字节数组，每个字节都有其唯一的地址（数组索引），从零开始。比如int4字节，long8字节。
  4. **处理器**：CPU，解释（或执行）存储在主存中指令的引擎，核心是一个一个字大小的存储设备（或寄存器），称为程序计数器（PC），在任何时刻，PC都指向主存中的某条机器语言指令(即含有该条指令的地址)，从通电开始，一直不断的执行PC指向的指令，再更新PC让它指向下一条指令

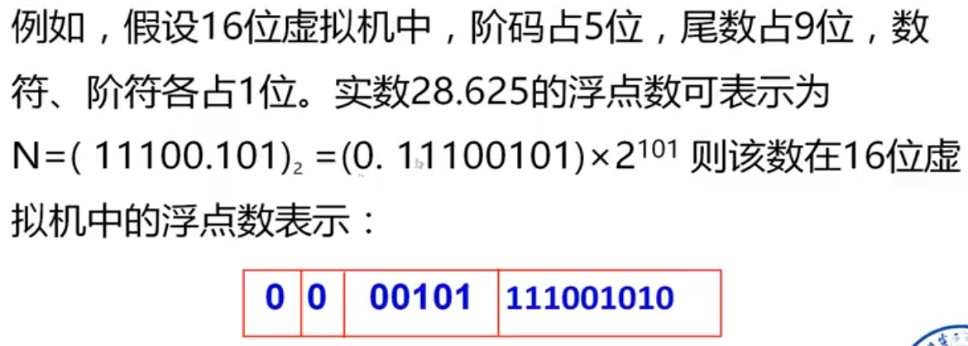


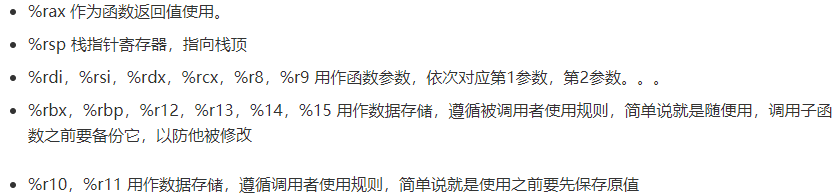
1. **高速缓存**
   1. 主存比硬盘快1000万倍，从寄存器读取数据比主存快100倍
   2. 加快处理器运行速度比加快主存的运行速度要容易和便宜的多
   3. 由此，高速缓存存储器出现了，存放处理器近期可能会需要的信息，L1容量可达数万字节，L2数十万到数百万，L2速度比L1慢5倍，但是仍然比访问主存快5-10倍
   4. L0就是寄存器
   5. 存储器层级结构的主要思想就是上一层作为低一层的高速缓存，比如寄存器是L1的高速缓存。
2. **操作系统**
   1. 两个基本功能：
      1. 防止硬件被失控的应用程序滥用；
      2. 向应用程序提供简单一致的机制来控制复杂而又大不相同的低级硬件设备
   2. 操作系统通过几个基本的抽象概念（进程，虚拟内存和文件）来实现这两个功能.
   3. **进程**：
      1. 无论在单核还是多核系统中，一个CPU看上去都像是在并发的执行多个进程，这是通过处理器在进程间切换来实现的，成为上下文切换。
      2. Shell和hello就是一个简单的例子，shell等待用户输入，用户输入hello回车后，操作系统保存shell进程的上下文，创建一个新的hello进程及其上下文，将控制转转给hello，hello终止后，恢复shell进程的上下文，将控制权传回给它，继续等待下一个输入
3. **信息的表示和存储**
   1. **信息存储**：大多数计算机使用8位的块或者字节作为最小的可寻址的内存单位，而不是访问内存中的单独的位。机器级程序将内存视为一个非常大的字节数组，成为**虚拟内存**，内存的每个字节由唯一数字标识，称为它的地址，所有可能地址的集合称为虚拟地址空间，目的是为程序提供一个看上去统一的字节数组。（**注**：这里好像解释了我之前的疑问，我一直在想汇编中会直接指定一个地址，不同的程序地址也是不一样的，那应用程序是怎么调用内存地址的呢，现在好像解释清楚了，应该是操作系统为程序提供虚拟内存，这样所有程序的使用的虚拟地址都是从0开始，这样就统一了，然后最后再由操作系统用一些方式映射到实际内存中，目前是这样猜测的，日后看看对不对吧）
   2. **进制**:
      1. 十进制转2进制，当值x是2的非负整数n次幂时，即x=pow(2,n)，二进制就是1后面跟n个0.
      2. 十进制转十六进制，当n表示成i+4j的形式，0<=i<=3，1 2 4 8对应0 1 2 3，后面跟j个十六进制的0，比如x=2048=pow(2,11)，n=11=3+4\*2，所以十六进制是0x800
   3. **字数据大小**
      1. 字长：指针数据的标称大小，虚拟地址是以这样的一个字来编码的，对一个字长为w位的机器，虚拟地址的范围为0-pow(2,w)-1，即pow(2,w)个字节.
      2. 大多数64位机器可以运行32位机器编译的程序，向后兼容,在用编译指令时不同的参数（32或64），32可以在两种位机器运行，64只能在64位，因此我们平时说的多少位的程序，区别在于是如何编译的，而不是运行时的机器类型。
      3. 指针在32位里都是4字节，在64位里是8字节
   4. **寻址和字节顺序**
      1. 对于跨越多字节的程序对象，以最小地址作为对象地址。比如int的地址为0x100，则4个字节分别为0x100,0x101,0x102,0x103
      2. 某些机器在内存中按照从最低到最高字节存储对象，成为小端法，相反，为大端法(**注**：刚开始理解错了，以为是地址是反的，其实是里面存的值是反的，如下图)
   5. **整数数据类型**
      1. 负数的范围比整数范围大1，知道如何表示负数，就知道为什么了
      2. W位的最大值是pow(2,w)-1，比如2位最大值是pow(2,2)-1=3
      3. 负数
         1. 第一位表示符号，1为负数
         2. 补码就是在原码的基础上，符号位不变，其余各位取反，最后+1，补码的目的是为了让计算机可以把减法当做加法，其实用反码也可以达到效果，但是会出现1-1=-0，为了消除掉0的负号，出现补码，算完之后反求原码就可以了， 并且10000000可以表示-128，所以补码的表示范围[-128,127]，原因是同余

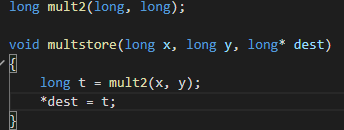
<https://www.zhihu.com/question/30395946/answer/47914507>

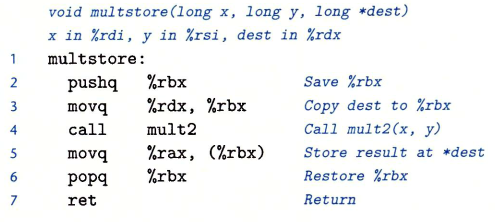
* + 1. 浮点数：
       1. 定点数：约定小数点的位置在符号位之后，即只能保存小于1的小数
       2. 浮点数：其中格式为把M化成定点数





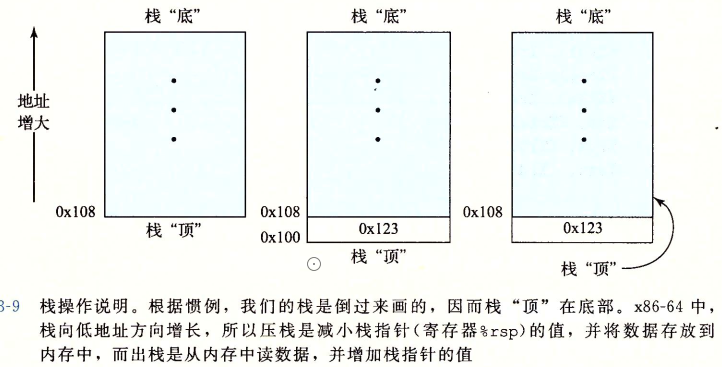
1. **程序的机器级表示**
   1. 程序编码（x86-64）
      1. 程序计数器（PC，用%rip表示），给出将要执行的下一条指令在内存中的地址
      2. 整数寄存器文件，包含16个命名的位置，分别存储64位的值
      3. 条件码寄存器，保存着最近执行的算术或逻辑指令的状态信息，用来实现条件，如if while
      4. 一组向量寄存器，存放数值
   2. 汇编语法
      1. 

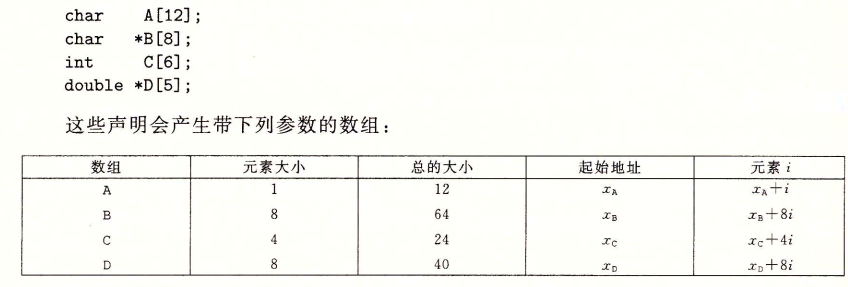
Mov 赋值前一个数到后面，这里看汇编代码其实解决了一些疑问.



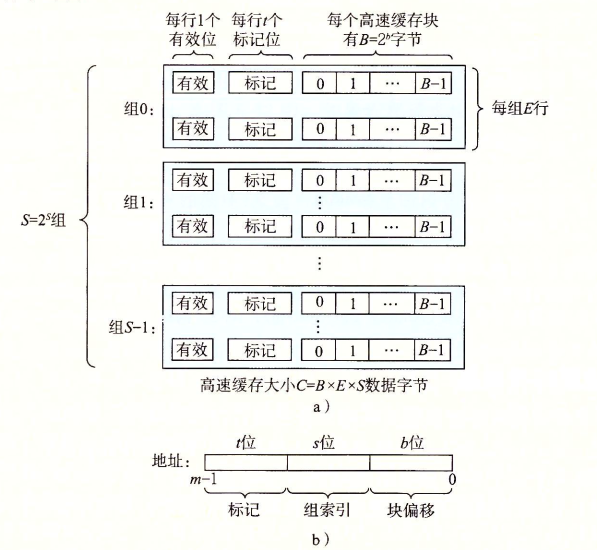
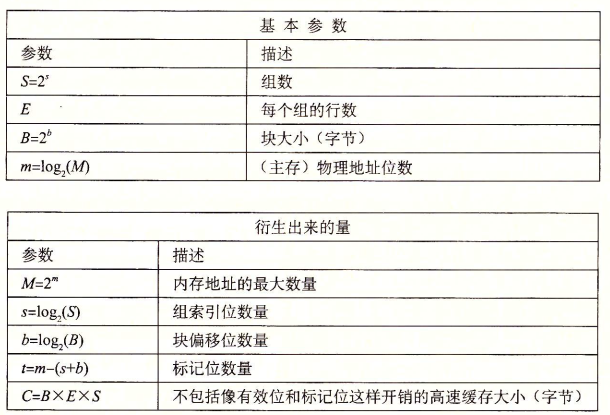
这里是先把寄存器rbx入栈，然后拷贝了dest的值到寄存器rbx，所以这里也解释了一切按值传递的原因，赋值后，出栈，完成函数调用

X86-64中，栈向低地址方向增长，所以压栈是减小栈指针的值，如图



* 1. 数组
     1. 

注意和本身的大小有关

1. 优化程序性能
   1. 缓存常用变量，比如循环时的size，这里编译器不会自动优化，因为它不能可靠的发现一个函数是否会有副作用
   2. 减少循环中的过程调用
   3. 5.6没太看懂，但是挺重要的，今天没心情看….
   4. 循环展开
   5. 提高并行性
2. 高速缓存存储器
   1. 
   2. 
   3. 这个图要好好解释一下，在写这个的时候我是理解的，不知道以后自己回来读的时候能不能看懂啊，加油啊未来的自己：
      1. 首先a是高速缓存的结构，有S组，每组有E行，每行有B字节的缓存块
      2. 主要的疑问是S=2^s，B=2^b，我一度以为印刷错误，分析了好久都想不通，然后才知道得和b结合
      3. b图是一个地址，用来表示如何这个地址如何能够表示出高速缓存的位置，地址是按位存储的，所以，最少用多少位可以表示S组呢，答案是2^s，即s位可以排列组合出S，可以想象一下，3位二进制，最多可以表示8位不同的数，因为2\*2\*2即2^3，以此类推，表示法大概就懂了
   4. 是预先知道组、块的大小，才根据地址去找对应高速缓存的位置，注意这个先后顺序，当读取一个地址时，会把其对应的块的所有地址都填充为读取的地址附近的地址，比如下图中的缓存结构，读地址0会把m[0]和m[1]放入组0的块中，当读地址9时，会把m[8]m[9]放入组0，并顶掉之前的

