★   寄存器命名原则    
  AT&T:   %eax                             Intel:   eax    
     
  ★源/目的操作数顺序    
  AT&T:   movl   %eax,%ebx         Intel:   mov   ebx,eax    
     
  ★常数/立即数的格式    
  AT&T:   movl   $\_value,%ebx   Intel:   mov   eax,\_value    
  把\_value的地址放入eax寄存器    
     
  AT&T:   movl   $0xd00d,%ebx   Intel:   mov   ebx,0xd00d    
     
  ★   操作数长度标识    
  AT&T:   movw   %ax,%bx             Intel:   mov   bx,ax    
     
  ★寻址方式    
  AT&T:       immed32(basepointer,indexpointer,indexscale)    
  Intel:     [basepointer   +   indexpointer\*indexscale   +   imm32)    
  Linux工作于保护模式下，用的是３２位线性地址，所以在计算地址时    
  不用考虑segment:offset的问题．上式中的地址应为：    
  imm32   +   basepointer   +   indexpointer\*indexscale    
     
  下面是一些例子：    
  ★直接寻址    
  AT&T:       \_booga　;   \_booga是一个全局的C变量    
  注意加上$是表示地址引用，不加是表示值引用．    
  注：对于局部变量，可以通过堆栈指针引用．    
     
  Intel:   [\_booga]    
     
  ★寄存器间接寻址    
  AT&T:       (%eax)    
  Intel:   [eax]    
     
  ★变址寻址    
  AT&T:       \_variable(%eax)    
  Intel:   [eax   +   \_variable]    
     
  AT&T:       \_array(,%eax,4)    
  Intel:     [eax\*4   +   \_array]    
  AT&T:       \_array(%ebx,%eax,8)    
  Intel:     [ebx   +   eax\*8   +   \_array]    
     
     
                                  二　基本的行内汇编    
     
  基本的行内汇编很简单，一般是按照下面的格式    
  asm("statements");    
  例如：asm("nop");   asm("cli");    
  asm　和　\_\_asm\_\_是完全一样的．    
  如果有多行汇编，则每一行都要加上　"   "    
  例如：    
  asm(         "pushl   %eax   "    
                  "movl   $0,%eax   "    
                  "popl   %eax");    
  实际上gcc在处理汇编时，是要把asm(...)的内容"打印"到汇编    
  文件中，所以格式控制字符是必要的．    
     
  再例如：    
  asm("movl   %eax,%ebx");    
  asm("xorl   %ebx,%edx");    
  asm("movl   $0,\_booga);    
     
  在上面的例子中，由于我们在行内汇编中改变了edx和ebx的值，但是    
  由于gcc的特殊的处理方法，即先形成汇编文件，再交给GAS去汇编，    
  所以GAS并不知道我们已经改变了edx和ebx的值，如果程序的上下文    
  需要edx或ebx作暂存，这样就会引起严重的后果．对于变量\_booga也    
  存在一样的问题．为了解决这个问题，就要用到扩展的行内汇编语法．    
                  三　扩展的行内汇编    
     
  扩展的行内汇编类似于Watcom.    
     
  基本的格式是：    
  asm   (   "statements"   :   output\_regs   :   input\_regs   :   clobbered\_regs);    
     
  clobbered\_regs指的是被改变的寄存器．    
  下面是一个例子(为方便起见，我使用全局变量）：    
  int   count=1;    
  int   value=1;    
  int   buf[10];    
  void   main()    
  {    
  asm(    
                  "cld     "    
                  "rep     "    
                  "stosl"    
                  :    
                  :   "c"   (count),   "a"   (value)   ,   "D"   (buf[0])    
                  :   "%ecx","%edi"   );    
  }    
  得到的主要汇编代码为：    
                  movl   count,%ecx    
                  movl   value,%eax    
                  movl   buf,%edi    
  #APP    
                  cld    
                  rep    
                  stosl    
  #NO\_APP    
  cld,rep,stos就不用多解释了．    
  这几条语句的功能是向buf中写上count个value值．    
  冒号后的语句指明输入，输出和被改变的寄存器．    
  通过冒号以后的语句，编译器就知道你的指令需要和改变哪些寄存器，    
  从而可以优化寄存器的分配．    
     
  其中符号"c"(count)指示要把count的值放入ecx寄存器    
  类似的还有：    
  a               eax    
  b               ebx    
  c               ecx    
  d               edx    
  S               esi    
  D               edi    
  I               常数值，(0   -   31)    
  q,r           动态分配的寄存器    
  g               eax,ebx,ecx,edx或内存变量    
  A               把eax和edx合成一个64位的寄存器(use   long   longs)    
     
  我们也可以让gcc自己选择合适的寄存器．    
  如下面的例子：    
  asm("leal   (%1,%1,4),%0"    
          :   "=r"   (x)    
          :   "0"   (x)   );    
  这段代码实现5\*x的快速乘法．    
  得到的主要汇编代码为：    
                  movl   x,%eax    
  #APP    
                  leal   (%eax,%eax,4),%eax    
  #NO\_APP    
                  movl   %eax,x    
  几点说明：    
  1.使用q指示编译器从eax,ebx,ecx,edx分配寄存器．    
      使用r指示编译器从eax,ebx,ecx,edx,esi,edi分配寄存器．    
  2.我们不必把编译器分配的寄存器放入改变的寄存器列表，因为寄存器    
  已经记住了它们．    
  3."="是标示输出寄存器，必须这样用．    
  4.数字%n的用法：    
      数字表示的寄存器是按照出现和从左到右的顺序映射到用"r"或"q"请求    
  的寄存器．如果我们要重用"r"或"q"请求的寄存器的话，就可以使用它们．    
  5.如果强制使用固定的寄存器的话，如不用%1,而用ebx,则    
  asm("leal   (%%ebx,%%ebx,4),%0"    
          :   "=r"   (x)    
          :   "0"   (x)   );    
  注意要使用两个%,因为一个%的语法已经被%n用掉了．