复习

刷题复习大纲--------->刷面经

进程状态以及切换方式

C++与C语言对比各个函数区别

函数指针指针函数

函数指针：void (\*p)(void)

指针函数：int \* fun(void);

数组指针：int (\*P)[10];

指针数组：int \*p[10];

构造数据类型

Strcpy memcpy区别

C++模式；

看完过一次大纲

项目ZigBee

汇编寄存器基本指令集

C++基本操作

一、引用的定义

引用是给另外一个变量起别名，所以引用不会分配内存空间。

引用的声明方法：类型标识符 &引用名=目标变量名;（如int &ptr = num;）

二、引用与指针的区别

1、指针是一个实体，需要分配内存空间。引用只是变量的别名，不需要分配内存空间。

2、引用在定义的时候必须进行初始化，并且不能够改变。指针在定义的时候不一定要初始化，并且指向的空间可变。（注：不能有引用的值不能为NULL）

3、有多级指针，但是没有多级引用，只能有一级引用。

4、指针和引用的自增运算结果不一样。（指针是指向下一个空间，引用时引用的变量值加1）

5、sizeof 引用得到的是所指向的变量（对象）的大小，而sizeof 指针得到的是指针本身的大小。

6、引用访问一个变量是直接访问，而指针访问一个变量是间接访问。

模板：

C++中为了区分同名类引入命名空间，using namespace std在同一个作用域中，不能有同名的类，但是不同的作用域当中可以有同名类的定义。

C++支持的四种类型强转：

1. const\_cast:去掉对象的const属性

2. static\_cast编译器认为可以支持的强转 安全性略高

3. reinterpret\_cast:类似C语言强转

4. dynamic\_castRTTI强制转换 run-time type information 必须要有虚函数

Class A

{

Virtual fun{}；

}

Class B ： public A

{

}

Class B b1;

Class A \*p = &b1;

Class B \* q = dynamic\_cast<Class B>(p);

1、输入迭代器：是只读迭代器，在每个被遍历的位置上只能读取一次。

2、输出迭代器：是只写迭代器，在每个被遍历的位置上只能被写一次。

3、前向迭代器：兼具输入和输出迭代器的能力，但是它可以对同一个位置重复进行读和写。但它不支持operator--，所以只能向前移动。

4、双向迭代器：很像前向迭代器，只是它向后移动和向前移动同样容易。

5、随机访问迭代器：有双向迭代器的所有功能。而且，它还提供了“迭代器算术”，即在一步内可以向前或向后跳跃任意位置。

容器：

函数定义前使用inline关键字，其目的是为了提高函数的执行效率

1、内联函数在编译时直接将函数体插入函数调用的地方。

2、inline只是一种请求，编译器不一定允许这种请求。

3、内联函数省去了普通函数调用时压栈，跳转和返回的开销。

4、内联函数在编译阶段不能生成符号，所以只有本文件可见，功能类似与static函数，但是static修饰的函数会产生本地的符号。内联函数和静态函数的定义一般可以放在头文件，但不能将内联函数的定义和内联函数的使用放在不同的源文件中。

5、内联只在release版本起作用，在debug版本不起作用，目的是为了方便调试。

6、与宏替换作用一样，只是内联发生在编译阶段、宏替换发生在预处理阶段，而且内联函数会做类型检查，但宏替换只是做简单地字符替换，并不会做类型检查。

C语言函数不能重载的原因？

C语言生成的函数符号由函数名决定，c++生成的函数符号由函数名和参数决定

静多态（编译链接）和动多态（运行）

静多态：重载、模板

动多态：继承和虚函数

重载的概念：函数参数不同，与返回值无关

全局作用域-----::

函数重载必须在相同作用域

带参宏 宏函数区别

1.函数调用时，先求出实参表达式的值，然后带入形参。而使用带参的宏只是进行简单的字符替换。

2.函数调用是在程序运行时处理的，分配临时的内存单元；而宏展开则是在编译时进行的，在展开时并不分配内存单元，不进行值的传递处理，也没有“返回值”的概念。

3.对函数中的实参和形参都要定义类型，二者的类型要求一致，如不一致，应进行类型转换；而宏不存在类型问题，宏名无类型，它的参数也无类型，只是一个符号代表，展开时带入指定的字符即可。宏定义时，字符串可以是任何类型的数据。

4.调用函数只可得到一个返回值，而用宏可以设法得到几个结果。

5.使用宏次数多时，宏展开后源程序长，因为每展开一次都使程序增长，而函数调用不使源程序变长。

6.宏替换不占运行时间，只占编译时间；而函数调用则占运行时间（分配单元、保留现场、值传递、返回）。

IP讲解https://www.cnblogs.com/red-code/p/7132023.html#\_label1\_2太详细了！！！！

IPV4地址：32位

IPV4被分为五大类：ABCDE

A类为：点分四组中的第一组地址范围为0~127的IP地址。已二进制来看就是“首位为0”

B类：128~191.二进制首位为10

C类：192~223.二进制首位为110

D类：224~239.二进制首位为1110

E类：240~255.二进制首位为1111

IPV6地址: 128位

IPv6地址长度是128位，

由8块（或8个字段）组成，每一块都包含四个16进制数，每块由冒号分隔。

有以下特点：

1、一个块中前导的0不必书写。

2、全0的块可以省略，并用符号::代替。

3、IPv6可以兼容IPv4地址，即可以用IPv6格式表示IPv4地址。

表示方式为：IPv6块值为ffff，其后面紧跟“点分四组”的格式。如：

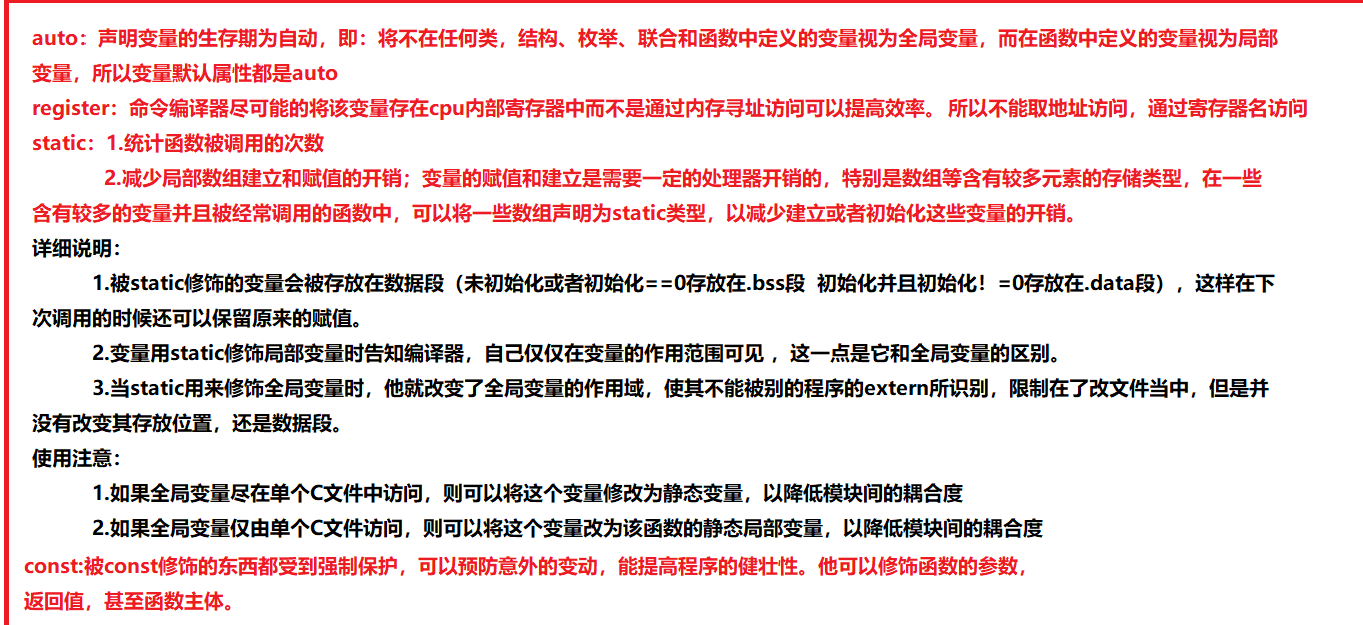
::ffff:10.0.0.1

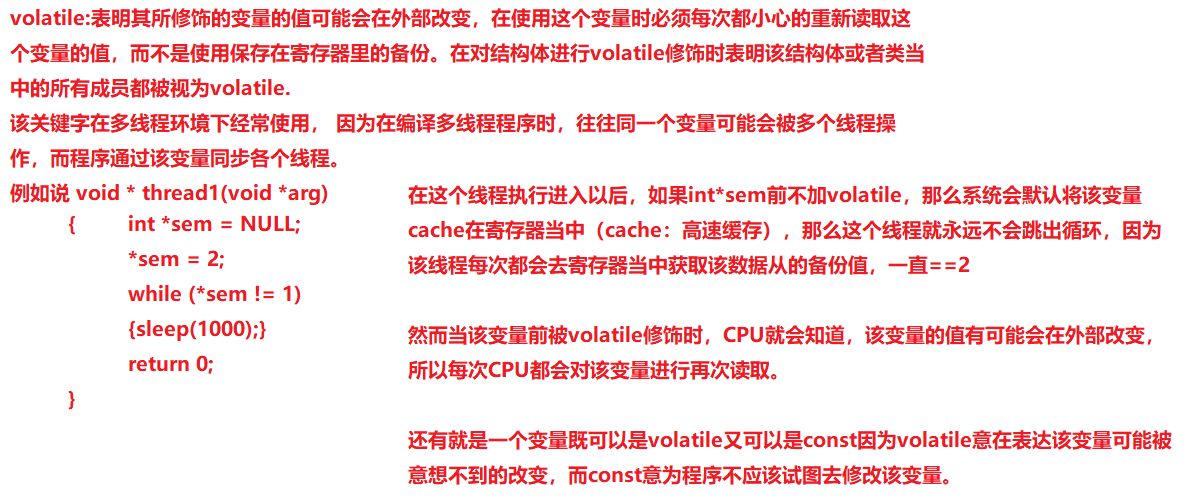
可以代表IPv4：10.0.0.1

4、IPv6的低32位通常采用点分四组（就是上面那样）的表示法。

MAC地址48位

===============================================================================





**禁止编译器优化-原地址读取**

==============================================================================

Typedef是用来为已有数据类型取类型别名，增加程序的可读性。

Define是预处理阶段宏定义指令。

1.执行时间不同

Typedef在编译阶段有效，有类型检查功能。

Define是发生在预处理阶段，也就是在编译阶段之前，所以只进行简单的字符的替换，不做类型检查。

2.功能不同

Typedef用来定义类型别名

Define取类型别名，还可以定义常量，变量，编译开关等。

3.作用域不同

Define没有作用域限制，只要在之前预定义过的宏，在以后的程序中都可以使用，而typedef有自己的作用域。

4.对指针进行修饰

Define如果代替的是指针型，那么其后如果有两个变量，那么只有第一个是指针型，第二个是基本数据类型。

Typedef具有扩展性，即一条语句后边的所有变量类型都与类型别名的类型保持一致。

===============================================================================

Const和Define

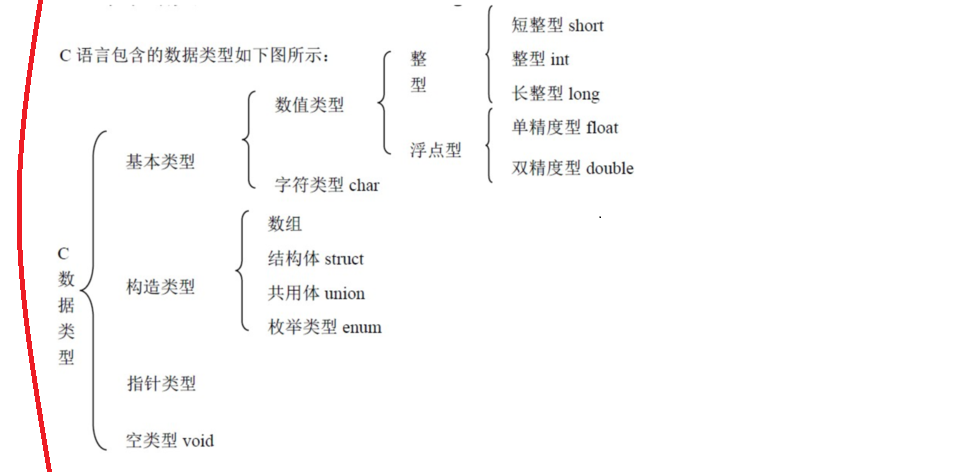
1. const在定义变量时必须指定数据类型，编译阶段处理，要做类型检测；const修饰的变量必须在定义时初始化。

define定义不带类型的常数，预处理阶段展开，只做简单的字符替换，不做类型检查。

1. 存储方式：

Define宏定义在预处理阶段没展开一次就要占用一次.text段空间

Const定义的变量在编译阶段分配内存，可以使堆（heep）也可以是栈（stack）



gbn sr区别

互斥锁不能实现线程的执行的先后顺序，锁一般用作多线程对全局变量进行操作过程中，保证全局变量的私有性

而信号量可以通过给不同资源初始化特定的资源量，则可以控线程的执行顺序。

位数组 链表 红黑树

1：什么是进程：

具有一定独立功能的程序在一个数据集上的一次动态执行过程，是操作系统进行资源分配的最小单元，是应用程序的载体。

组成：程序、数据集、PCB

程序：指定进程完成什么样的功能，是控制进程执行的指令集

数据集：程序执行时所需要的数据和工作区

PCB:包含进程的描述和控制信息，是进程存在的标志

2：什么是线程：

程序执行中一个单一的顺序控制流程,是程序执行的最小单位，是处理器进行调度的最小单位。

进程线程的区别：（线程的优点）

1.一般来说线程是操作系统进行调度的最小单元，而进程是操作系统进行资源分配的最小单元

2.一个进程可以包含多个线程，线程是进程中代码的不同执行路线

3.进程之间是相互独立的，但是一个进程中的各个线程共享该进程的代码段，数据段，堆，以及一些进程级别的资源：比如打开的文件，信号等等，某个进程中的线程对其他的进程是不可见得。线程独享资源：栈，线程id，当前指令指针：pc寄存器

4，线程的上下文切换速度要比进程快的多，因为同一个进程中的多个线程共享该进程的文本段、数据段、堆、以及进程级别的资源，但是每个进程都有自己独有的4G虚拟地址空间，所以线程间的切换只完成在同一个4G虚拟地址空间中的切换，而进程间的切换完成的是不同虚拟地址空间之间的切换过程。

5、多线程适用于占用时间比较少的，占用资源量少的服务，比如说是一般的网页请求的短链接类服务，因为它们处理这一类事件轻便，快捷，服务结束后销毁线程，开销小。

6、进程内部的资源独立，避免了线程操作导致的数据不一致在加上进程本身相对稳定的，所以对于多进程而言，适合长时间资源量较大的链接服务，比如说文件下载这种需要长时间保持连接的服务。进程间切换少，适合长时间对外提供稳定的服务。

线程的缺点：

1、线程之间共享的资源太多，所以对于线程间的共享变量要进行同步和加锁控制，很麻烦。

2、一个线程的崩溃可能导致整个进程的崩溃。

3、线程的总个数是受到限制的 ulimit -a -》max user processes

一个进程做多可以打开200多个线程

而一般操作系统可以开好多进程

多进程的优点：

1、进程之间是相互独立的，子进程的崩溃不会影响主进程的运行

2、增加CPU个数后，性能会得到非常大的提升

3、可以减少加锁或者信号量的开销

进程间通信方式：

1、无名管道

2、有名管道

3、signal信号

4、消息队列

5、共享内存

6、信号灯

7、网络socket

8、封装、继承、多态

9、模式：单例？？？

======================================================================

IO5种模型：

1、阻塞IO：进程一直阻塞，直到数据拷贝完成。

应用程序调用一个IO函数,导致应用程序阻塞，等数据准备好。如果数据没准备好，一直等待，直到数据准备好为止。

2、非阻塞IO：非阻塞IO是通过进程反复调用IO函数（多次调用立即返回）；并在数据拷贝过程中，进程是阻塞的；

3、IO复用（select 、 poll、 epoll）：

4、信号驱动IO：首先允许套接口进行信号驱动IO，并安装一个信号处理函数，进程并不阻塞，当数据准备好时，进程会收到一个SIGIO信号，可以在信号处理函数中调用IO操作函数处理数据。

5、异步IO：当代码执行到IO操作时，CPU只需要发出IO指令，并不用等待IO结果

然后去执行另一段代码，当IO返回结果时，再通知CPU进行处理。

时间复杂度；1，log2n，n，n log2n ，n的平方，n的三次方，2的n次方，n!

IO复用：

所有的IO复用底层都是采用回调函数的机制来实现的，首先都是在内核中建立存储文件描述符的一张表，在注册回调函数，当有IO事件发生时，回调函数告诉进程，来处理。

select：是将文件描述符存在一个位数组当中，交给内核进行监听，当有事件发生时，会同过回调函数通知进程，将所有的文件描述符从内核当态加载到用户态，然后对整个位数组进行一个遍历，来处理事件。

由于它将文件描述符存在位数组当中，所以能监听的文件描述符是有限的。并且对其进行增删改查的时间复杂度都是O（n）

poll：和select本质上机制是相同的，但是它是将文件描述符存放在一个链表当中。

由于它是将文件描述符存在一个链表当中，所以存入的个数可以说是没有限制，并且在链表的插入删除时间复杂度可以接近Log2n；但是他和select一样，全部从内核态加载到用户态，全部遍历查找。

epoll：

epoll在执行之前首先会调用epollcreate在内核创建一个内核事件表，内核事件表的底层是一个红黑树，红黑树其实就是一个高度平衡的BST树，他用红黑树来存放要监听的文件描述符，然后调用epollctl将文件描述符添加到内核时间表中，同时为每一个文件描述符创建属于其自己的回调函数，并且可以设置文件描述符的模式：可以设置文件描述符的存放属性的，ET模式和LT模式；

调用epoll\_wait然后交给内核监听该内核事件表，有IO事件发生时，文件描述符就会触发自己的回调函数，将自己的文件描述符加载到rdlist表中.然后他会将rdlist复制一份，放在txlist，因为在处理事件时，有可能是处理完成和未完成两种状态，所以要对txlist中的文件描述符进行检测，如果文件描述符是LT模式，如果该文件描述符没有被处理，他就会将该文件描述符重新拷贝到rdlist中，提示进程再次处理该文件描述符，但是如果文件描述符是ET模式的话，就算这个文件描述符没有被处理，也不会将它拷贝到rdlist中，也就是说ET模式的文件描述符，只给进程一次处理机会，如果没有处理，就跳过该文件描述符。

Tcp和Udp的区别：

1.Tcp面向连接（如打电话之前要先拨号），UDP是无连接的，即发送数据之前是不用建立连接的

2、TCP提供的是可靠服务，也就是说，通过TCP连接传输的数据，无差错，无丢失，不乱序，不重复

而UDP提供的是一种尽力而为式的服务，即不保证可靠交付。

3、TCP面向字节流，将数据看成一连串无结构的字节流，UDP是面向报文的，UDP没有拥塞控制，因此网络发生拥塞不会影响源主机的发送速率，这对实时很有用，如实时视频会议等

4、每一条TCP连接只能是点对点的，UDP支持一对一，一对多，多对一等模式

5、TCP首部开销比较大20字节，UDP首部开销小，只有8个字节，报头最大长度为60字节。

6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠的全双工通信方式

7、UDP程序结构简单

回答此问题还有一种很好的回答方式，对比TCP UDP的报头信息来回答

TCP：

源端口16位 | 目的端口16位

32位序号

32位确认号

4位头部| 6位保留 |U|A|P|R|S|F| 16位窗口大小

长度 | |R| C|S|S|Y| I |

| |G|K|H|T|N|N|

16位校验和 | 16位紧急指针

选项和填充

数据

（1）16位源端口 以及目16位目的端口

（2）32位序号：序号是本报文段发送的数据组的第一个字节的序号。在TCP传送的流中，每一个字节一个序号。e.g.一个报文段的序号为300，此报文段数据部分共有100字节，则下一个报文段的序号为400。

A:实现数据传送的有序性 B:保证重复数据报被丢弃

（3）32位确认号：对A端发送的数据进行确认并回馈的序号。（给32位序号值+1返回回去）

即ACK，指明下一个期待收到的字节序号，表明该序号之前的所有数据已经正确无误的收到。确认号只有当ACK

标志为1时才有效。比如建立连接时，SYN报文的ACK标志位为0。 保证了数据能够有效地到达接收端

（4）4位头部长度：报头不包含任何任选字段则长度为20字节，4位首部长度字段所能表示的最大值为1111，转化为10进制为15，15\*32/8 = 60，故报头最大长度为60字节。首部长度也叫数据偏移，是因为首部长度实际上指示了数据区在报文段中的起始偏移值。

（5）6位标志位：

URG：表示紧急指针是否有效

ACK：表示确认号是否有效

PSH：通知尽快读取数据

RST：通知重新建立连接

SYN：请求建立连接

FIN：请求断开连接

（6）16位窗口大小：告诉发送方接收方的缓存大小，控制发送数据的速度或流量

（7）16位校验和：由发送端填充，接收端对TCP报文段在传输过程中是否损坏。检验不仅包括报头部分，还包括数据部分。

（8）16位紧急指针：一个正的偏移量（相当于当前序号的偏移），紧急指针是发送端向接收端发送紧急数据的方法。

UDP报头信息：

---------------------------------------------------------

源端口 | 目的端口

---------------------------------------------------------

用户数据长度| 校验和

---------------------------------------------------------

用户数据

---------------------------------------------------------

（1）用户数据长度：包含报头和数据部分的总长度

（2）校验和：UDP协议不做错误纠正，发现错误丢掉发出警告

各自的优缺点：

1、TCP优点：

可靠、稳定、面向连接、流量控制、重传、拥塞控制

缺点：

慢、效率低、资源占用量多

应用场景：对效率要求相对低，但对数据准确性要求相对高，或者有链接的场景

：FTP HTTP（对数据准确性要求较高, 速度可以相对慢） pop SMTP（对互数据准确性要求高，非紧急应用）

远程登录（对数据准确性有一定要求）。

2、UDP优点：

快、UDP没有拥塞控制，所以网络出现拥塞不会影响源主机的发送速率、开销小

缺点：

不可靠、不稳定、易丢包

应用场景：一般应用与即时通信；聊天（对数据准确性和丢包要求相对较低，但速度必须快），在线视频，网络电话，视频会议等等，对实时性要求比较高，偶尔有丢包可以忍受。

TCP:

TCP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt(); \* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、开启监听，用函数listen()；

　　5、接收客户端上来的连接，用函数accept()；

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

　　8、关闭监听；

TCP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置要连接的对方的IP地址和端口等属性；

　　5、连接服务器，用函数connect()；

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

UDP:

与之对应的UDP编程步骤要简单许多，分别如下：

　　UDP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、循环接收数据，用函数recvfrom();

　　5、关闭网络连接；

UDP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置对方的IP地址和端口等属性;

　　5、发送数据，用函数sendto();

　　6、关闭网络连接；

UDP应用场景：

1.面向数据报方式

2.网络数据大多为短消息

3.拥有大量Client

4.对数据安全性无特殊要求

5.网络负担非常重，但对响应速度要求高

如果udp丢包了，怎么办？udp乱序了，怎么办？

    udp丢包和乱序都是很正常的，因为它本来就是不安全的。

    udp丢包是因为udp没有流量控制，所以发送速度可能要比接受速度快，这样一来，接收端还没来得及处理完数据，发送端就已经发完了，剩下没进入缓冲区的那一部分就丢掉。

解决方法：

1.降低发送的速度，可能网络性能导致的丢包

2.设置setsockopt将缓冲区放大一点。

3.增加一个响应报文，收到响应报文后，再进行继续发送，有点类似有tcp的ack

4，至于乱序问题，对数据报进行编号

TCP三次握手四次挥手：

三次握手：

1. 一开始，服务器处于listen状态，等待客户连接，客户端开始连接，发送自己的SYN，处于SYS\_SENT。
2. 服务器接收到请求，accept处于SYS\_RCVD状态，收到，回复一个ACK，SYN类型的响应报文。
3. 客户端再次进行确认，确认后发送自己的ACK，三次握手结束，二者都处于ESTABLESHED

为什么是三次握手，为什么不能是2次或者4次？

 两次握手会发生什么情况 ？因为TCP是一个可靠得传输协议，它自身带有应答机制和重传机制，如果只有两次握手，客户端向服务器发起连接，只有服务器一端进行ACK回复，服务器在没有收到客户端啊的ACK之前，都认为是ACK丢失，会

再次重传，这样的情况只会让系统资源不断消耗

四次握手是没有必要的，因为三次握手之后，两边都已确定对方能接收到自己的数据，所以，四次，五次握手都是没有任何意义的。

四次挥手：

1. 想结束的一方先发送自己的结束请求报文FIN, 自己进入FIN\_WAIT状态
2. 另一方接收到关闭请求，回复一个ACK确认关闭响应报文，进入CLOSE\_WAIT状态

/\*在这两个步骤之间服务器会进入一个等待状态,因为也许客户端这边数据发送完成了，但是作为服务器，还有数据要发给客户端，所以服务器紧接着还要发送一个FIN报文\*/

1. 紧接着该方发送自己的结束报文即FIN报文，处于LAST\_ACK，等待最后一次响应
2. 发起结束端对FIN进行最后响应，四次挥手结束。

5、整个四次挥手阶段进入time\_wait状态。

Time\_wait状态的作用：因为到第四次挥手时，客户端的最后一个ACK并不是绝对能可靠地到服务器，所以客户端必须在这里等待一段时间，以便于下次服务器FIN到来时，能够再次响应ACK；time\_wait的等待时间一般是2MSL。如果时间到了，没有收到FIN则才能彻底完成四次挥手。（MSL：报文最大生存周期）

===============================================================================HTTP协议：

请求报文：



请求方法：Get

--------------------------------------------------------------------------------------------------------------------------------

GET /562f25980001b1b106000338.jpg HTTP/1.1

Host img.mukewang.com

User-Agent Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari/537.36

Accept image/webp,image/\*,\*/\*;q=0.8

Referer http://www.imooc.com/

Accept-Encoding gzip, deflate, sdch

Accept-Language zh-CN,zh;q=0.8

--------------------------------------------------------------------------------------------------------------------------------

POST：

POST / HTTP1.1

Host:www.wrox.com

User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)

Content-Type:application/x-www-form-urlencoded

Content-Length:40

Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley

第一部分：请求行，第一行明了是post请求，以及http1.1版本。

第二部分：请求头部，第二行至第六行。

第三部分：空行，第七行的空行。

第四部分：请求数据，第八行。

--------------------------------------------------------------------------------------------------------------------------------

响应报文：

HTTP/1.1 200 OK

Date: Fri, 22 May 2009 06:07:21 GMT

Content-Type: text/html; charset=UTF-8

<html>

<head></head>

<body>

<!--body goes here-->

</body>

</html>

第一部分：状态行，由HTTP协议版本号， 状态码， 状态消息 三部分组成。

第一行为状态行，（HTTP/1.1）表明HTTP版本为1.1版本，状态码为200，状态消息为（ok）

第二部分：消息报头，用来说明客户端要使用的一些附加信息

第二行和第三行为消息报头，

Date:生成响应的日期和时间；Content-Type:指定了MIME类型的HTML(text/html),编码类型是UTF-8

第三部分：空行，消息报头后面的空行是必须的

第四部分：响应正文，服务器返回给客户端的文本信息。

空行后面的html部分为响应正文。

HTTP协议中的请求方法：  
GET：请求指定的页面，返回实体主体

POST：向指定资源提交数据进行处理请求，例如提交表单或者上传文件，数据被包含在使用主体中。

PUT：从客户端向服务器传送的数据取代指定的文档内容。上传

HEAD：类似GET请求，只不过返回的响应中没有具体内容，用于获取报头

DELETE：删除文件

GET方法和POST的区别：

1. GET提交，请求数据会附在URL字段中包体中；POST提交：将数据放在HTTP报文的包体中，所以GET提交的数据会在地址栏中显示出来，而POST提交，不会显示
2. GET不同的浏览器URL字段的长度会有限制，POST传输数据理论上不会受到限制。
3. POST的安全性要比GET高，比如GET提交数据，用户名和密码明文显示在URL字段，并且可能被浏览器缓存，可以通过历史记录就可以看到用户名和密码。

GET 方法

请注意，查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

/test/demo\_form.asp?name1=value1&name2=value2

有关 GET 请求的其他一些注释：

GET 请求可被缓存

GET 请求保留在浏览器历史记录中

GET 请求可被收藏为书签

GET 请求不应在处理敏感数据时使用

GET 请求有长度限制

GET 请求只应当用于取回数据

POST 方法

请注意，查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

POST /test/demo\_form.asp HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

有关 POST 请求的其他一些注释：

POST 请求不会被缓存

POST 请求不会保留在浏览器历史记录中

POST 不能被收藏为书签

POST 请求对数据长度没有要求

常见状态码：

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

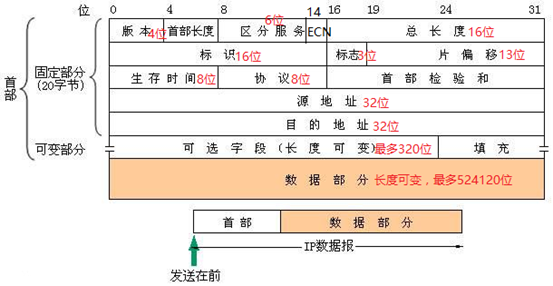
403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

IP协议



IP首部行：

4位版本：IP数据报的版本号：IPV4 IPV6

4位首部长度：5位最大长度：15\*32/8 = 60字节

6位区分服务：3位优先级（由数据链路层确定，定义了8个优先级）优先级越高字段越优先传输。3位状态信息：都是置‘1’生效：D（低延时）T（吞吐量）R（可靠性）

2位ECN：用于为数据报标记“拥塞标识符”如果ECN置位，表示发生拥塞，接收端告诉发送端放慢发送速率。

16位总长度：指定IP数据报总长度，通过该字段和“首部长度”字段可以确定数据部分的起始地址。

16位标识、3位标志、16位片偏移：后边详讲.

标识符：

主机将数据报分片后，在发送前，会给每一个分片数据报一个ID值，放在16位的标识符字段中。

这个ID值可以用来识别哪些分片是属于同一个数据报的，方便重组。

标志：

标志字段在IP报头中占3位，

第1位作为保留，置0;

第2位，分段，有两个不同的取值：该位置0，表示可以分段;该位置1，表示不能分段;

第3位，更多分段，同样有两个取值：该位置0，表示这是数据流中的最后一个分段，该位置1，表示数据流未完，后续还有分段，当一个数据报没有分段时，则该位置0，表示这是唯一的一个分段。

当目的主机接收到一个IP数据报时，会首先查看该数据报的标识符，并且检查标志位的第3位是置0或置1，以确定是否还有更多的分段，如果还有后续报文，接收主机则将接收到的报文放在缓存直到接收完所有具有相同标识符的数据报，然后再进行重组。

偏移量：

就像之前说过的，各个IP分片数据报在发送到目的主机时可能是无序的，所以就需要“偏移量”字段来指明“该分片在原数据报中的位置顺序”。

发送主机对第一个数据报的偏移量置为0，而后续的分片数据报的偏移量则以网络的MTU大小赋值。

如：

假设：网络接口MTU大小为1400字节，要传输的数据报为3800字节。

那么，将要传输的数据报分为3片即可（3800=1400+1400+1000）

偏移量的计算方法为：已经“装载”好的分片字节数/8（偏移量就是某片在原分组的相对位置，所以8个字节作为偏移单位。）

分片1偏移量为：0（0=0/8，因为该偏移量之前没有装载过任何分片，自然也就是0除以8了）

分片2偏移量为：175（175=1400/8，由于分片1已经装载好了1400字节，所以此分片的位置就在1400字节之后，也就用1400除以8了）

分片3偏移量为：350（350=2800/8，目前已经装好了两个分片也就是2800字节已经装载完了，那么分片3自然就跟在2800字节之后了，也就是用2800除以8）

8位生存时间：设置数据报可经过的路由器数量的上限，由发送方填充一个数，在每经过一个路由器，数字减一，直到为0数据报丢弃，使用ICMP消息通知对方。

8位协议：区分数据报中信息的类型：1：ICMP 2：IGMP 4：IP 6：TCP 17：UDP 45：IDRP 46：RSVP 47：GRE 54：NHRP 88：IGRP 89：OSPF

16位首部校验和：只检查首部，不检查数据

可选字段：可写可不写

32位源地址：ip

32目的地址：ip

大小端判断过程中，int-》char发生截断，char占的是低地址 共用体中char占的永远也是低地址

时间片即CPU分配给各个程序的时间，每个线程被分配一个时间段，称作它的时间片，即该进程允许运行的时间，使各个程序从表面上看是同时进行的。如果在时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。而不会造成CPU资源浪费。在宏观上：我们可以同时打开多个应用程序，每个程序并行不悖，同时运行。但在微观上：由于只有一个CPU，一次只能处理程序要求的一部分，如何处理公平，一种方法就是引入时间片，每个程序轮流执行

所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何 context switch （切 [1] 换到另一个线程）。

重载：同一个类当中的同名不同参的多个函数会发生重载

隐藏：在基类与派生类之间，如果派生类中有与基类函数同名的函数，在派生类中，对基类函数时不可见的，但是可以用Base：：fun（）来调用基类被隐藏函数

覆盖：隐藏函数的一种特殊情况，即基类函数被vactor修饰、并且发生在不同的两个类当中、并且必须与基类函数同名并且同参。

堆和栈的区别

2）栈内存是由系统分配，系统释放的；以函数为单位进行栈内存分配，函数栈帧，局部变量，形参变量等都存放在栈内存上。堆内存是由用户自己分配的，C语言用malloc/free进行分配释放，C++用new/delete进行分配释放，由于堆需要用户自己管理，因此堆内存很容易造成内存泄露，而栈内存不会。

3）栈的内存分配释放速度快效率高，内存都是连续的；堆内存的分配释放相对于栈来说效率低一些，内存不一定连续，容易产生内存碎片，但是灵活性高。

4）栈是由高地址向低地址扩展的连续内存，栈的大小一般为2M或者10M（大家的redhat系统可以用ulimit -s命令来查看，是10M）；堆是由低地址向高地址扩展的非连续内存，堆的大小影响的因素比较多，和系统虚拟内存的大小有关系。

（1）堆向高内存地址生长，栈向低内存地址生长。

（2）申请后系统的响应：

　　栈：只要栈的剩余空间大于所申请的空间，系统将为程序提供内存，否则将报异常提示栈溢出。

　　堆：首先应该知道操作系统有一个记录内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请的空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样代码中的delete或free语句就能够正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会将多余的那部分重新放入空闲链表中。

（3）申请的大小限制不同：

　　栈：在windows下，栈是向低地址扩展的数据结构，是一块连续的内存区域，栈顶的地址和栈的最大容量是系统预先规定好的，能从栈获得的空间较小。

　　堆：堆是向高地址扩展的数据结构，是不连续的内存区域，这是由于系统是由链表在存储空闲内存地址，自然堆就是不连续的内存区域，且链表的遍历也是从低地址向高地址遍历的，堆得大小受限于计算机系统的有效虚拟内存空间，由此空间，堆获得的空间比较灵活，也比较大。

（4）申请的效率不同：

　　栈：栈由系统自动分配，速度快，但是程序员无法控制。

　　堆：堆是有程序员自己分配，速度较慢，容易产生碎片，不过用起来方便。

（5）堆和栈的存储内容不同：

　　栈：在函数调用时，先进栈的是函数的各个参数，然后是主调函数中下一条执行指令的地址，最后是主调函数的栈底地址ebp，在大多数的C编译器中，参数是从右往左入栈的。当本次函数调用结束后，局部变量先出栈，然后栈底指针值出栈（赋给ebp）并使ebp指向主函数的栈底，下一条指令的地址值出栈（赋给pc寄存器）并使栈顶指针esp指向主函数的栈顶，最后将各个参数从左到右依次出栈。

堆：一般是在堆得头部用一个字节存放堆得大小，具体内容由程序员安排。

总结

　　数据量较小时，推荐使用栈空间申请，即直接定义数组

　　数据量稍大或者不确定时，推荐使用堆空间内存，即使用malloc或者new动态申请，因为栈空间常常会有大小的限定，当栈空间耗尽时，栈溢出会导致程序崩溃

　　当数据量超大的，建议重新审阅算法或者使用文件存储

　　栈空间与子函数，递归与栈溢出

当一个子函数被调用时，子函数的数据及代码都会被装入栈中，因为栈空间通常会有大小限制，如果子函数太多时，就会有栈溢出的风险。所以当程序员考虑使用递归函数解决问题时，应当考虑到栈溢出的风险。建议学会使用将递归函数写成非递归函数的方法

18 21 25

默认PC=0；0指向IROM，上电之后PC=0，IROM先执行，拷贝启动设备中的前24K到IRAM中执行。

Linux系统移植流程：  
1.搭建交叉编译开发环境

2.BootLoader的制作移植

3.kernel的配置、编译、移植

4.根文件系统的制作

1.搭建交叉编译开发环境：

因为嵌入式开发针对的是不同平台，不同的平台都有自己不同的体系结构和指令集，要在开发主机上开发出能够在目标机上运行的程序，就必须要有交叉编译环境。ARM MIPS POWEPC等

并且通常嵌入式中目标机的性能都不是很高，要完成linux kernel的编译非常慢，如果有了交叉编译环境，就可以在性能较高的开发机上完成这些操作。

交叉编译的硬件组成主要有以下部分：

1.开发主机 2、目标机（开发板）3、链接介质：1串口线 2 USB线 3 网线

配置nfs服务：实现网络文件的挂载，通过网络构建一个目标机与开发机共享文件夹，这样我们就可以开发在开发机，运行在目标机

2.dnw（usb传输）配置：实现裸机开发阶段，开发机向目标机发送编译好的内核

2、BootLoader的选择与制作：

使用uboot工程制作BootLoader工程

1、首先先编译uboot：

1.先确定平台 make tiny4412\_config

2.编译uboot

2、将制作好的BootLoader烧录到SD卡中

BootLoader制作完成

将开发板调至SD卡启动

3、制作内核：

1、将下载好的linux-3.5解压

2、修改架构以及平台信息

3、内核裁剪：

1、内核所拥有的功能全部存放在各个Kconfig

2、我们要通过make menuconfig来选择自己所需要的内核功能（y,m,n）修改的是各个功能模块的Makefile文件是否参加内核的编译

3、我们所选择的功能被存放在顶层目录的.config文件中

4、通过选择性编译来实现内核的裁剪

5、make 编译内核生成裁剪后的内核镜像保存在arch/arm/boot文件中，将内核镜像通过usb下载到开发板，bootm启动内核镜像。

4、根文件系统制作没有制作

Arm的启动流程：

1、上电后IROM首先开始运行（因为ARM默认上电后pc指针指向0地址，启动代码存放在0地址）。

2、程序拷贝启动设备中的前24K代码到IRAM上执行即加载BootLoader到内存当中运行。

BootLoader作用：

1、为操作系统内核运行作环境准备，初始化cpu、存储、内存、串口。

2、加载内核，引导内核，向内核传参（告诉内核根文件系统的位置和方式）。

3、kernel运行，挂载根文件系统，至此ARM启动完成。

ARM体系结构：

Arm公司设计的精简指令集处理器架构

Armv1 -> Armv8

Exyons4412属于armv7

Arm7， arm9， arm11

Cortex M 工业控制

Cortex R 实时性

Cortex A 消费电子（微处理器）

A7， 8， 9， 15， 35， 53， 72， 73， 75， 76

Arm核心：4核

1、寄存器多

2、指令长度定长 32位

3、寄存器都是32位

4、大部分指令单周期

5、处理交互数据只能来自寄存器，不能直接来自内存

Arm共有37个32位寄存器，其中包括31个通用寄存器，6个状态寄存器。

通用寄存器：r0---r15

R0-R12普通寄存器

R13：sp 栈寄存器 在程序发生跳转时可以将寄存器值保存到sp所指向的空间

R14：lr 链接寄存器 保存程序跳转前pc寄存器的值，跳转结束返回主程序

R15：pc 程序计数器 对其进行写入操作：cpu跳转到写入的地址去执行代码

对其进行读取操作：当前正在执行指令的下下条代码

状态寄存器：常用

Cpsr：当前程序状态寄存器

N：31位 1表示结果为负数 0表示正数

Z：30位 1表示结果为0 0表示结果非0

C：29位 加法VEIF

Spsr：保存Cpsr的数值

ARM有7种工作模式，cortex\_a9 9种每种工作模式都有寄存器（公用和私有）。

非特权：

User 用户：大部分应用程序的工作模式

特权：

Sys 系统：和user模式使用同一组寄存器的特权模式

异常模式：

Undefined mode 未定义模式：未识别指令的工作模式

Abort mode 中止模式：预取失败，访问受保护空间时cpu的工作模式

Supervisor mode 管理模式：上电复位cpu的工作模式，软件中断cpu的工作模式

IRQ mode 中断模式：irq中断的cpu工作模式（硬件中断）

FIQ mode 快速中断模式：fiq中断时CPU的工作模式。

Monitor mode 安全模式

Hyp mode 虚拟化模式

异常发生时 CPU自动切换

在特权模式（管理模式）修改cpsr寄存器

arm的异常处理机制

arm异常信号：7种

复位异常 Supervisor mode

未定义异常 Undefined mode

软件中断（svc） Supervisor mode

预取中止 Abort mode

数据中止 Abort mode

irq中断 IRQ mode

fiq中断 FIQ mode

当异常发生时，cpu会进行异常区分，自动跳转到异常向量表固定位置执行代码异常向量表：异常事件处理方法集合

异常向量表基地址：高位 0xffff0000低位 0x00000000

armv7 设置在任意位置

arm协处理器 完成电路管理

16个协处理器 CP0-CP15

汇编指令：机器码的助记符

数据处理指令

跳转指令

内存操作指令

块拷贝指令

栈操作指令

特殊寄存器操作指令

协处理器指令

指令格式：

MOV{S}<c> <Rd>, #<const>

{S}该指令可以被写成movs，根据运算结果更改cpsr寄存器的NZCV

<c>指令可以条件执行

<Rd>目标寄存器

#<const>立即数

伪指令：

1、nop 空操作

2、ldr r0， =0x12345678 //可以将任意32位数传递给目标寄存器

3、lar r0， =fun //将fun的链接地址给r0

4、adr r0， fun 将fun的地址给r0

立即数：必须以 #开始

必须是一个8位以内的数，经过循环左移或者右移偶数位形成的数值。

Mvn取反传输

Mvn{S}<C><Rd>， #<const>

Mvn r0， #0 == r0 = ~0

0, r1 == r0 = ~r1

Add r0, r1, #3

Add r0, r1, r2

运算指令

add 加法指令

add r0, r1, #3 === r0 = r1 + 3

add r0, r1, r2 === r0 = r1 + r2

sub 减法指令

sub r0, r1, #3 === r0 = r1-3

sub r0, r1, r2 === r0 = r1-r2

sub r0, r0, #3

sub r0, #3 === r = r0-3

rsb 反减

rsb r0, r1, #3 === r0 = 3-r1

rsb r0, r1, r2 === r0 = r2-r1

mul

mul r0, r1, r2 === r0 = r1\*r2

64位数据运算（高32位运算）

adc 带进位加法

adc r0, r1, r2 === r0 = r1+r2+C

sbc 带借位减法

sbc r0, r1, r2 === r0 = r1-r2-!C

64位运算：

adds

adc

subs

sbc

移位操作

MVN{S}<c> <Rd>, <Rm>{, <shift>}

<Rm>{, <shift>} 表示 <Rm>先进行移位操作，后参与mvn的运算

LSL #<n>

LSR #<n>

ASR #<n>

ROR #<n>

mvn r0, r1, lsl #3

lsl 逻辑左移 高位丢失，低位补0

lsl r0, r1, #3 === r0 = r1<<3

lsl r0, r1, r2 === r0 = r1<<r2

lsr 逻辑右移 低位丢失，高位补0

lsr r0, r1, #3 === r0 = r1>>3

lsr r0, r1, r2 === r0 = r1>>r2

asr 算术右移 移位操作按照逻辑右移，符号位不变，空位补符号位

ror 循环右移， 低位移动到高位

位操作

and 按位与

and r0, r1, #3 === r0= r1&3

and r0, r1, r2 === r0= r1&r2 可以进行清0操作， 可以进行判断

orr 按位或

orr r0, r1, #3 === r0= r1|3

orr r0, r1, r2 === r0= r1|r2 可以实现置位操作

eor 按位异或

eor r0, r1, #3 === r0= r1^3

eor r0, r1, r2 === r0= r1^r2 判断相等

bic 位清除 清除第一操作数中，第二操作数中为1的那些位

bic r0, r1, #1 === r0 = r1 & ~(1)

bic r0, r1, r2 === r0 = r1 & ~(r2)

条件码：

ne 判断不等

eq 判断相等

le 小于等于

lt 小于

gt 大于

ge 大于等与

比较指令

cmp 比较 所有条件

cmp r0, #3 利用r0-3的结果修改CPSR寄存器的NZCV

cmp r0, r1 利用r0-r1的结果修改CPSR寄存器的NZCV

cmn 负数比较

cmn r0, #3 利用r0+3的结果修改CPSR寄存器的NZCV

cmn r0, r1 利用r0+r1的结果修改CPSR寄存器的NZCV

teq 测试相等 eq ne

teq r0, #3 利用r0^3的结果修改CPSR寄存器的NZCV

teq r0, r1 利用r0^r1的结果修改CPSR寄存器的NZCV

tst 位测试 测试第一操作数中，第二操作数为1的位是0 ne测试的位不全为0 eq表示测试的位全是0

tst r0, #3 利用r0&3的结果修改CPSR寄存器的NZCV

tst r0, r1 利用r0&r1的结果修改CPSR寄存器的NZCV

地址相关代码:程序运行地址和程序链接地址必须一致

地址无关代码:程序运行地址和程序链接地址不必一致

ldr r0, =fun 将标号表示的地址传递给r0

跳转指令：

长跳转 ，写pc

mov pc , #0xff000000

ldr pc, =0x12345678

ldr pc, =fun

短跳转(相对跳转)，基于当前pc前后前后32M跳转（找标号），到标号位置处执行代码

b fun 只跳转

bl fun 保存返回的跳转（使用lr保存返回地 mov lr， pc）

bx

标号：地址的助记符，标号不占任何空间

load/store 指令完成内存和寄存器数值的交互

ldr 从内存加载一个字内容到目标寄存器

str 保存一个寄存器内容到指定内存空间

ldr r0, fun 将fun标号地址处的内容赋给r0

ldr r0, [r1] === r0 = \*r1

str r0, [r1] === \*r1 = r0

! 表示基地址更新

前索引

ldr r1, [r0, #4] === r1 = \*(r0+4) r0=r0

str r1, [r0, #4] === \*(r0+4)=r1 r0=r0

后索引

str r0, [r1], #4 ; \*r1=r0 r1=r1+4

ldr r3, [r1], #4 ;r3=\*r1 r1=r1+4

基址变址索引

ldr r1, [r0, #4]! === r1 = \*(r0+4) r0=r0+4

str r1, [r0, #4]! === \*(r0+4)=r1 r0=r0+4

ldrb/strb 8位操作

ldrh/strh 16位操作

ldr/str 32位

ldrd/strd 64位操作

小编号寄存器对应低地址空间

块拷贝指令：

ldm 从内存加载连续多个字内容到多个寄存器

ldm r0, {r1,r2,r3} === r1=\*r0 r2=\*(r0+4) r3=\*(r0+8) r0=r0

stm 保存多个寄存器数值到连续内存空间

stm r0, {r1,r2,r3} === \*r0=r1 \*(r0+4)=r2 \*(r0+8)=r3

ldmia/stmia 先内存操作，后地址+4

ldmib/stmib 先地址+4，后内存操作

ldmda/stmda 先内存操作，后地址-4

ldmdb/stmdb 先地址-4，后内存操作

入栈：stmdb sp!, {r0-r12}

出栈：ldmia sp!, {r0-r12}

满减栈（默认使用） sp指向的空间不能直接使用，栈的生长方向向下

满增栈 sp指向的空间不能直接使用，栈的生长方向向上

空减栈 sp指向的空间可以直接使用，栈的生长方向向下

空增栈 sp指向的空间可以直接使用，栈的生长方向向上

栈操作指令：

ldmfd/stmfd 满减栈

ldmfa/stmfa 满增栈

ldmed/stmed 空减栈

ldmea/stmea 空增栈

stmfd sp!, {r0-r12}

ldmfd sp!, {r0-r12}^ ^表示模式恢复

push {r0-r12}

pop {r0-r12}

特殊寄存器操作指令：

cpsr寄存器操作：

msr cpsr, r0 === 将r0内容写入cpsr

mrs r0, cpsr === 读取cpsr数值到r0寄存器

协处理器操作指令：

cp15协处理器

cp15协处理器有16个32位寄存器

mrc p15, 0, r0, c0, c0, 0 将cp15协处理器的c0寄存器数值读取到r0

mcr p15, 0, r0, c1, c0, 0 将r0数值写入cp15协处理器的c1寄存器

I2C：