任务安排

任务

```
任务：先总结知识点，后每个知识点找两篇博客。

```

知识点

```
知识点--1：typedef类型的用法， 1.https://blog.csdn.net/sruru/article/details/7916296/（教学详细）
                                2.https://blog.csdn.net/helaisun/article/details/54845891（例程丰富）

知识点--2：函数指针类型的用法   1.http://c.biancheng.net/view/228.html(教学详细)
                                2.https://blog.csdn.net/modi000/article/details/105980765（例程丰富）

知识点--3：二重指针与指针数组   1.https://blog.csdn.net/Zorro721/article/details/107304645（教学详细）
                                2//zhidao.baidu.com/question/2205614253334274268.html（教学详细）

知识点--4：DMA                  1.https://blog.csdn.net/wuyongpeng0912/article/details/46634931
                                2.https://www.jianshu.com/p/b19a7c46f465

知识点--5：芯片启动流程         1.https://blog.csdn.net/qq_46359697/article/details/115255840

2.https://blog.csdn.net/weixin_43593698/article/details/108303376

知识点--6：sscanf函数的使用     1.https://baike.baidu.com/item/sscanf/10551550?fr=aladdin
                                2.https://www.runoob.com/cprogramming/c-function-sscanf.html

知识点--7：uint32_t的溢出       1.https://wenku.baidu.com/view/804921cb5df7ba0d4a7302768e9951e79b8969ae.html

知识点--8：解耦与耦合           1.https://blog.csdn.net/nanhuaibeian/article/details/106059409
                                2.https://www.cnblogs.com/zhjblogs/p/14221560.html

知识点--9：查询法和中断法区别   1https://bbs.csdn.net/topics/370263037

```

总结

7.c > show(FUNC2, int, int *)

```c
#include <stdio.h>
int   inc(int   a)
{
    return(++a);
}

int   multi(int*a,int*b,int*c)
{
    return(*c=*a**b);
}

typedef   int(FUNC1)(int);
typedef   int(FUNC2)(int*,int*,int*);

void   show(FUNC2   fun,int   arg1,   int*arg2)
{
    FUNC1   *p = &inc;
    int   temp   =p(arg1);
    fun(&temp,&arg1,   arg2);
    printf( "%d\n ",*arg2);
}

int main()
{
    int   a;
    show(multi,10,&a);
}
```

FUNC1和FUNC2定义为返回值为int的，参数为（int)的整型和 返回值为int，参数为（int*， int*， int*)的类型；

_____

_____

```c
int Max(int, int);  //函数声明
int(*p)(int, int);  //定义一个函数指针

int main(void)
{
    int a, b, c;
    p = Max;  //把函数Max赋给指针变量p, 使p指向Max函数
    a = 1; b = 2;
    c = (*p)(a, b);  //通过函数指针调用Max函数
    system("pause");
    return 0;
}
int Max(int x, int y)  //定义Max函数
{
    if (x > y)
        return x;
    else
        return y;
}
```

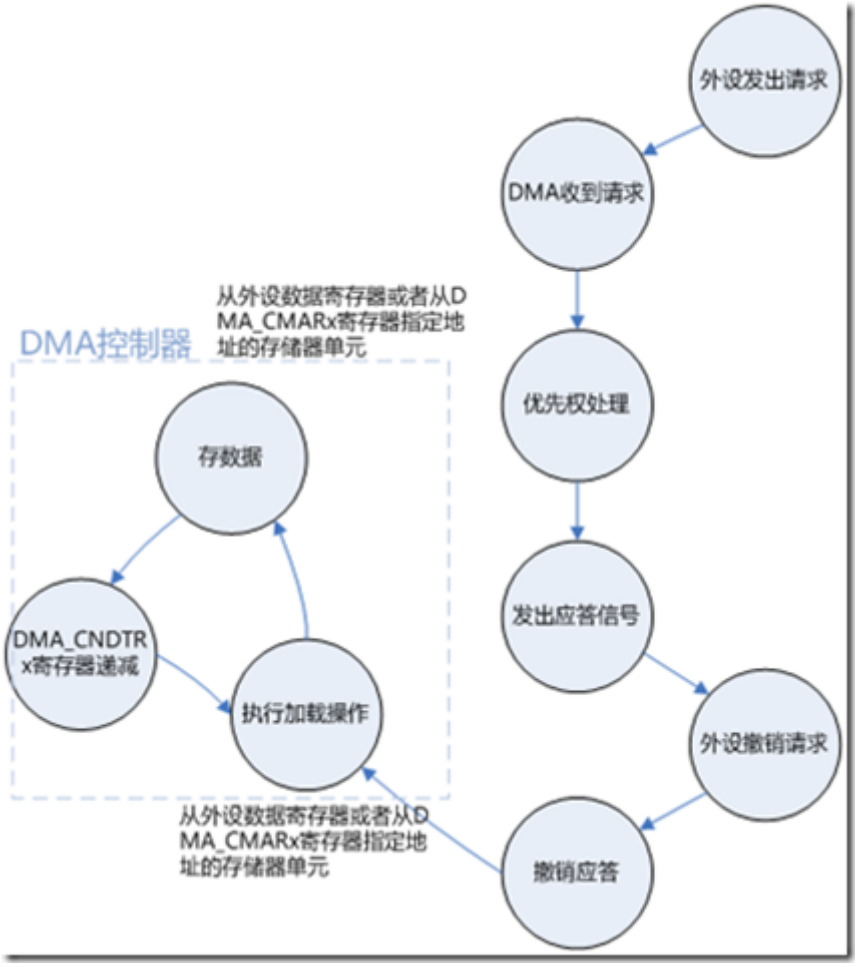
2.函数指针类型的用法

函数名Max就是调用这个函数的入口地址。就如同定义一个数组，这个数组名就是这个数组的首地址。
既然是地址，就可以使用一个指针来指向它。
对应于int Max(int x, int y)，定义的函数指针如下：
与函数声明类似，int(*p)中的int代表函数的返回值类型为int；
括号内的两个int代表参数列表；
这里的*代表声明的是函数指针；
不可将括号忘掉，否则表示声明的是一个函数，返回值类型为int*；

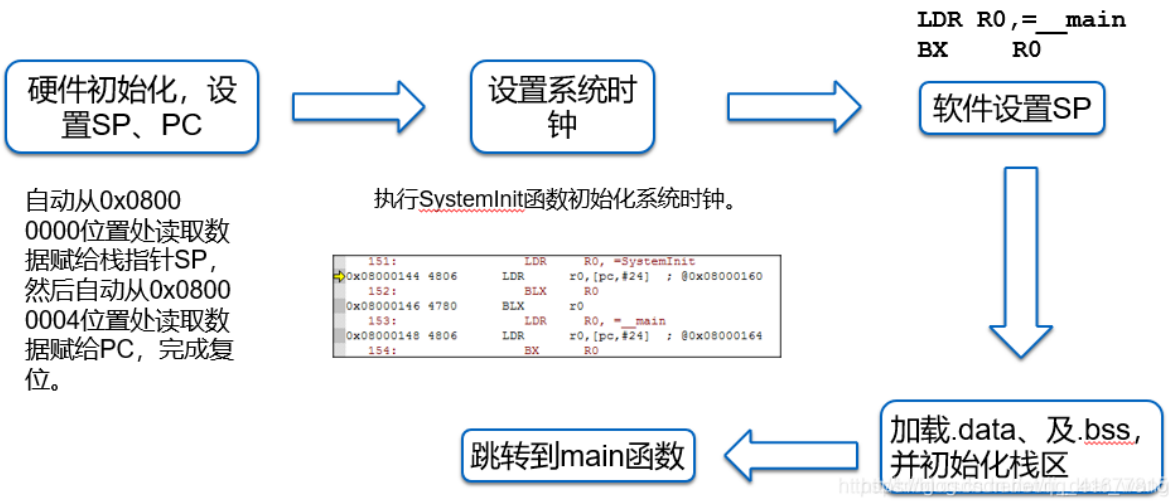
```c
char* ordersplit_result[10] = {0};
splitstr(g_buffer, ordersplit_result, "+=,\r\n");
```

```
int splitstr(const char* in, char** splitresult, const char* split_mark_table) {
  static char buf[300];
  strcpy(buf, in);
  int off = 0;

  *splitresult = buf;
  off++;

  for (size_t i = 0; buf[i] != '\0'; i++) {
    if (strcontain2(buf[i], split_mark_table)) {
      buf[i] = '\0';

      splitresult[off] = &buf[i+1];
      printf("%s\r\n",splitresult[off]);
      off++;
    }
  }
  return off - 1;
```

```
//3.二重指针与指针数组

    char buf[100];
    char** spliteresult1;
    char* ordersplit_result[10];
    spliteresult1 = ordersplit_result;
     spliteresult1[0] = ordersplit_result[0];
    ordersplit_result[0] = "first";
    ordersplit_result[1] = "second";

//    **spiteresult1是指向 ordersplit_result指针数组中的内容，*ordersplit_result是指向数组的内容的首地址，ordersplit_result是二重指针地址，*ordersplit_result是二重指针地址里面的值。
//    下图中，将ordersplit_result的地址赋给splitresult，*spliterresult是buf的首地址，spliteresult[off]中存放的是每个分割完后下一个字符串开头的首地址。
```




```
//4.DMA
Direct Memory Access（存储器直接访问）。这是指一种高速的数据传输操作，允许在外部设备和存储器之间直接读写数据。整个数据传输操作在一个称为"DMA控制器"的控制下进行的。CPU除了在数据传输开始和结束时做一点处理外（开始和结束时候要做中断处理），在传输过程中CPU可以进行其他的工作（前提是未设置停止CPU访问）。这样，在大部分时间里，CPU和输入输出都处于并行操作。因此，使整个计算机系统的效率大大提高”。

DMA传送方式是让存储器与外设、或外设与外设之间直接交换数据，不需要经过CPU的累加器中转，减少了这个中间环节，并且内存地址的修改、传送完毕的结束报告都是由硬件电路实现的，因此大大地提高了数据的传输速度。一个DMA传送只需要执行一个DMA周期，相当于一个总线读写周期。

DMA是在专门的硬件（DMA）控制下，实现高速外设和主存储器之间自动成批交换数据尽量减少CPU干预的输入/输出操作方式。

```



5.芯片启动流程
①上电后硬件设置SP、跳转到 Reset_Hander

②设置系统时钟(SystemInit)

③软件设置SP

④加载.data、.bss，并初始化栈区(__main)

⑤跳转到C文件的main函数

值得注意的是：Keil编译完成后：

Code：代表程序代码段

RO_DATA:代表只读数据段

RW_DATA：代表已经初始化全局数据

ZI_DATA：代表未初始化全局数据

由于程序在 FLASH 中直接通过总线进行访问，程序运行在 FLASH 上，而可更改的数据存于 SRAM 中，故：

RO_SIZE = Code + RO_DATA（占用 Flash ）

RW_DATA = RW_DATA + ZI_DATA（占用 SRAM）

ROM_SIZE = Code + RO_DATA + RW_DATA (烧写到 FLASH 大小空间)

针对 ZI 数据，是不存 FLASH 中，直接在 SRAM 中初始化为 0

6.sscanf的使用

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
  int day, year;
  char weekday[20], month[20], dtm[100];
```

```
   strcpy( dtm, "Saturday March 25 1989" );
   sscanf( dtm, "%s %s %d  %d", weekday, month, &day, &year );

   printf("%s %d, %d = %s\n", month, day, year, weekday );

   return(0);
}
```

声明

下面是 sscanf() 函数的声明。

```
int sscanf(const char *str, const char *format, ...)
```

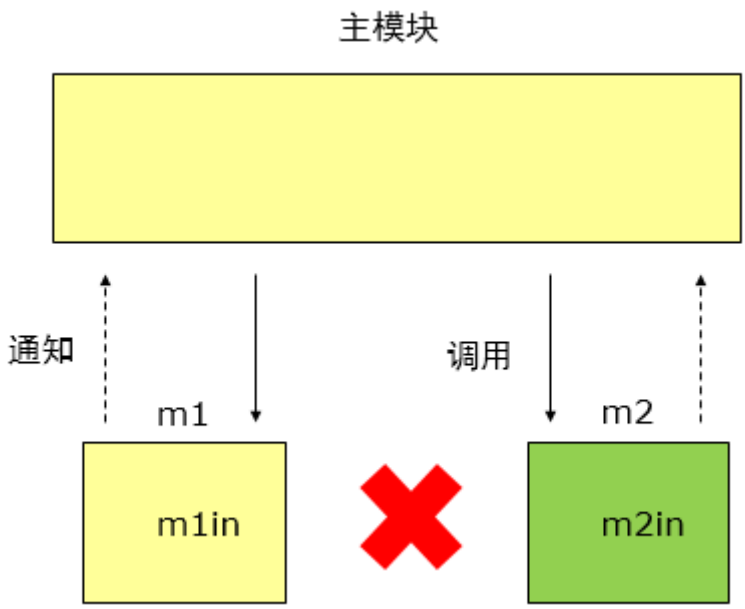
参数

str -- 这是 C 字符串，是函数检索数据的源。

format -- 这是 C 字符串，包含了以下各项中的一个或多个：空格字符、非空格字符 和 format 说明符。

format 说明符形式为 [=%[*][width][modifiers]type=]，具体讲解如下：

参数	描述
*	这是一个可选的星号，表示数据是从流 stream 中读取的，但是可以被忽视，即它不存储在对应的参数中。
width	这指定了在当前读取操作中读取的最大字符数。
modifiers	为对应的附加参数所指向的数据指定一个不同于整型（针对 d、i 和 n）、无符号整型（针对 o、u 和 x）或浮点型（针对 e、f 和 g）的大小：h：短整型（针对 d、i 和 n），或无符号短整型（针对 o、u 和 x）l：长整型（针对 d、i 和 n），或无符号长整型（针对 o、u 和 x），或双精度型（针对 e、f 和 g）L：长双精度型（针对 e、f 和 g）
type	一个字符，指定了要被读取的数据类型以及数据读取方式。具体参见下一个表格。



7.解耦与耦合
什么是耦合、解耦
一、耦合（可以称为关联性）
1、耦合是指两个或两个以上的体系或两种运动形式间通过相互作用而彼此影响以至联合起来的现象。
2、在软件工程中，对象之间的耦合度就是对象之间的依赖性。对象之间的耦合越高，维护成本越高，因此对象的设计应使类和构件之间的耦合最小。
3、分类：有软硬件之间的耦合，还有软件各模块之间的耦合。耦合性是程序结构中各个模块之间相互关联的度量。它取决于各个模块之间的接口的复杂程度、调用模块的方式以及哪些信息通过接口。
二、解耦
1、解耦，字面意思就是解除耦合关系。
2、在软件工程中，降低耦合度即可以理解为解耦，模块间有依赖关系必然存在耦合，理论上的绝对零耦合是做不到的，但可以通过一些现有的方法将耦合度降至最低。
3、设计的核心思想：尽可能减少代码耦合，如果发现代码耦合，就要采取解耦技术。让数据模型，业务逻辑和视图显示三层之间彼此降低耦合，把关联依赖降到最低，而不至于牵一发而动全身。原则就是A功能的代码不要写在B的功能代码中，如果两者之间需要交互，可以通过接口，通过消息，甚至可以引入框架，但总之就是不要直接交叉写。
4、观察者模式：观察者模式存在的意义就是「解耦」，它使观察者和被观察者的逻辑不再搅在一起，而是彼此独立、互不依赖。比如网易新闻的夜间模式，当用户切换成夜间模式之后，被观察者会通知所有的观察者「设置改变了，大家快蒙上遮罩吧」。QQ消息推送来了之后，既要在通知栏上弹个推送，又要在桌面上标个小红点，也是观察者与被观察者的巧妙配合

7.查询法和中断法区别
中断程序在程序开始定义中断入口地址，初始化中必须打开中断允许位，程序运行时不用判断溢出状态位，溢出后硬件清零；

查询方式在程序运行时必须判断溢出状态位，溢出后须软件清零。

查询方式，就是在主函数里面不停循环，查询端口状态，明显其弊端在于响应速度，在处理事件多，处理流程复杂，函数嵌套执行的情况下，由于处理不过来容易丢失事件。

举个例子，在电话用户接入系统里面，一个单片机管理1个电话端口的摘挂机，执行周期要求8ms，用查询的方式足够了，但是当电话增加到16个，用查询方式，效果就差了，曾出现过电话响起的时(12个电话齐呼)，拿起话筒，电话还在振铃，明显处理不过来。

这个时候，有两个办法，一个采用中断方式，另一个采用更高效的CPU，明显前者只需要修改软件，后者需要增加硬件成本，还延长开发时间。