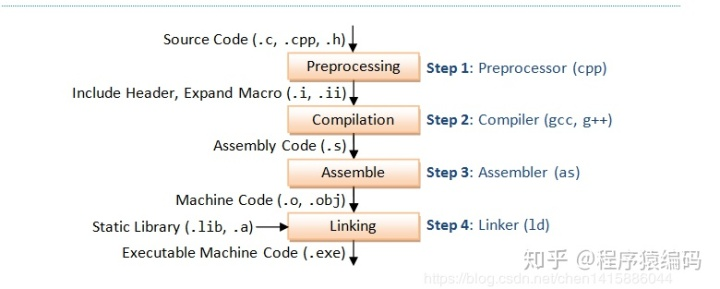
gcc 交叉编译

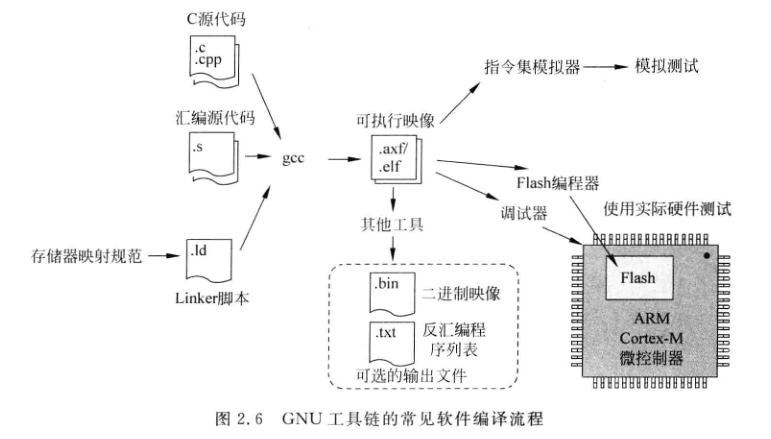
# 1编译过程

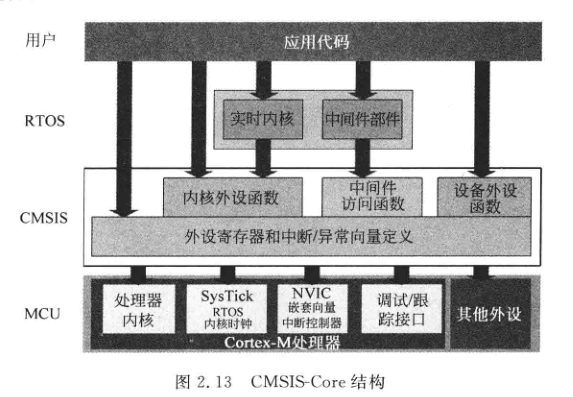


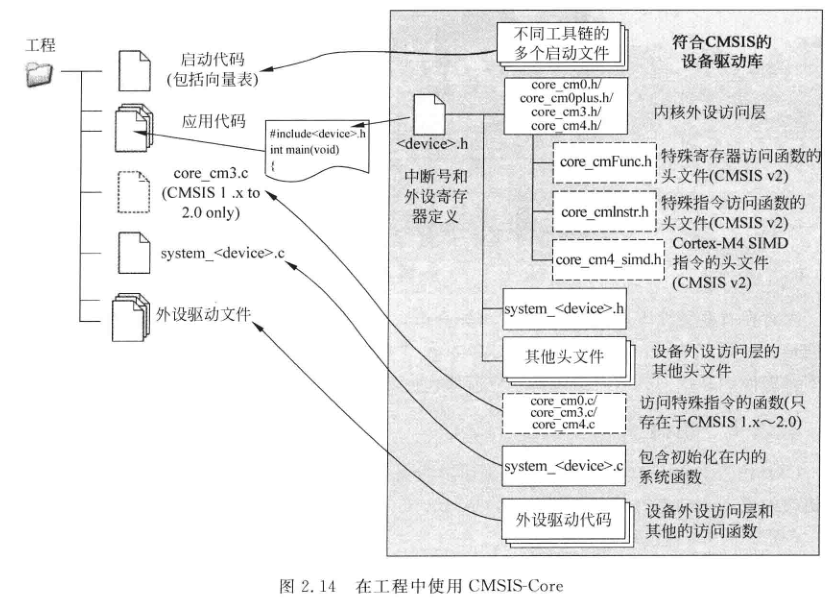
Gcc 编译包含编译+链接两个过程，共有4个步骤：

1. 预处理(Preprocessing)：处理预处理指令，如#include，#define等。
2. 编译(Compilation)：将c代码转换为汇编语言。
3. 汇编(Assemble)：将汇编语言转换为二进制机器码。
4. 链接(Linking)：将各目标文件、库文件、启动文件进行链接，生成可执行文件。

补充的图：







# 2交叉编译

所谓"交叉编译（Cross\_Compile）"，是指编译源代码的平台和执行源代码编译后程序的平台是两个不同的平台。比如，在Intel x86架构/Linux（Ubuntu）平台下、使用交叉编译工具链生成的可执行文件，在ARM架构/Linux下运行。

交叉编译是相对复杂的，必须考虑如下几个问题：

* CPU架构：比如ARM，x86，MIPS等等；
* 字节序：大端（big-endian）和小端（little-endian）；
* 浮点数的支持；
* 应用程序二进制接口（Application Binary Interface,ABI）；

为什么要使用交叉编译呢？主要有两个原因：

* 交叉编译的目标系统一般都是内存较小、显示设备简陋甚至没有，没有能力在其上进行本地编译；
* 有能力进行源代码编译的平台CPU架构或操作系统与目标平台不同；

# 3项目脚本



## 预处理+编译+汇编

### 3.1.1文件

用到的文件如下：

* .c文件：(a)bench/tb\_WUKONG\_4T4R.c，(b)tb\_\*/source/gcc/\*.c，以及

(c) /projects/wukong\_m4f/common/gcc\_dependence/CMSIS/4.2.0/Device/ARM/ARMCM4/Source/system\_ARMCM4.c

* .S文件：

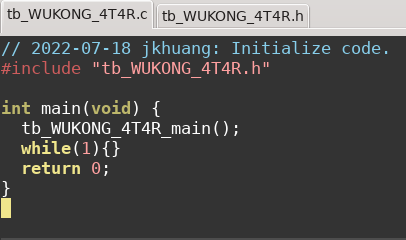
(d) /projects/wukong\_m4f/common/gcc\_dependence/CMSIS/4.2.0/Device/ARM/ARMCM4/Source/GCC/startup\_ARMCM4\_ap\_cp.S

* .h头文件：

(e) bench/ap\_cp.h

* 此外，还包含c的基础库文件（stdio.h，stdint.h等）以及(f) CMSIS库文件。

**(a)** **bench/tb\_WUKONG\_4T4R.c：**

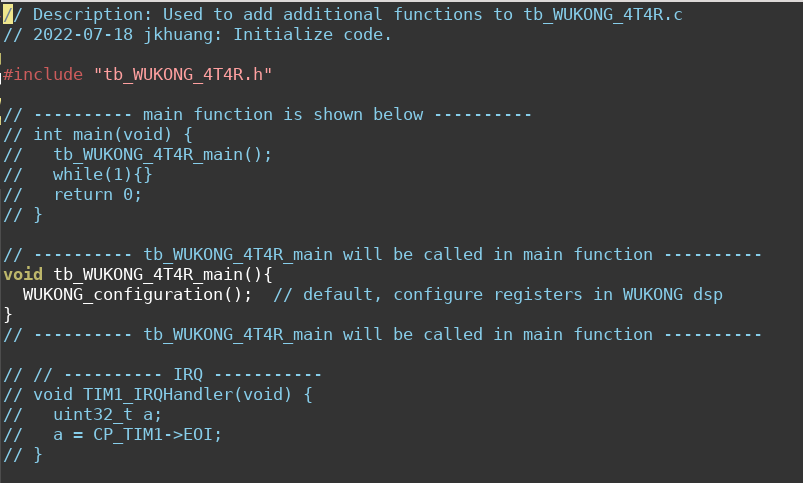


调用tb\_WUKONG\_4T4R\_main()后进入死循环

**(b)** **tb\_\*/source/gcc/\*.c：**

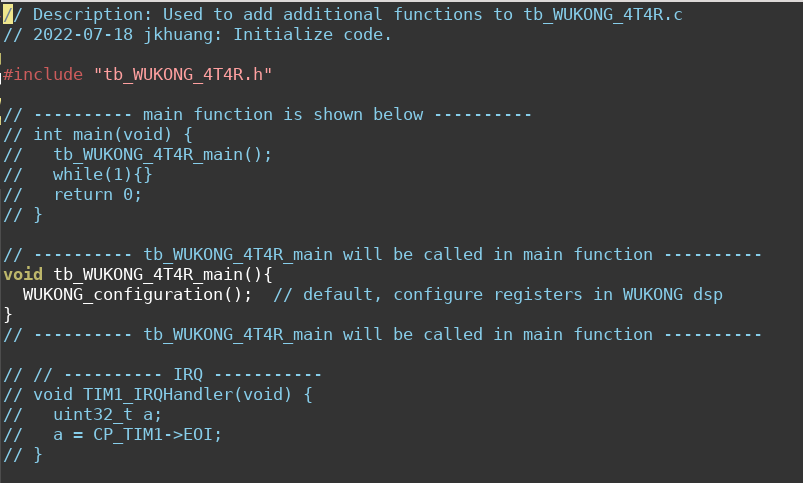
包含tb\_WUKONG\_4T4R\_main.c和WUKONG\_configuration.c，还可以自己添加其他文件。

tb\_WUKONG\_4T4R\_main.c：



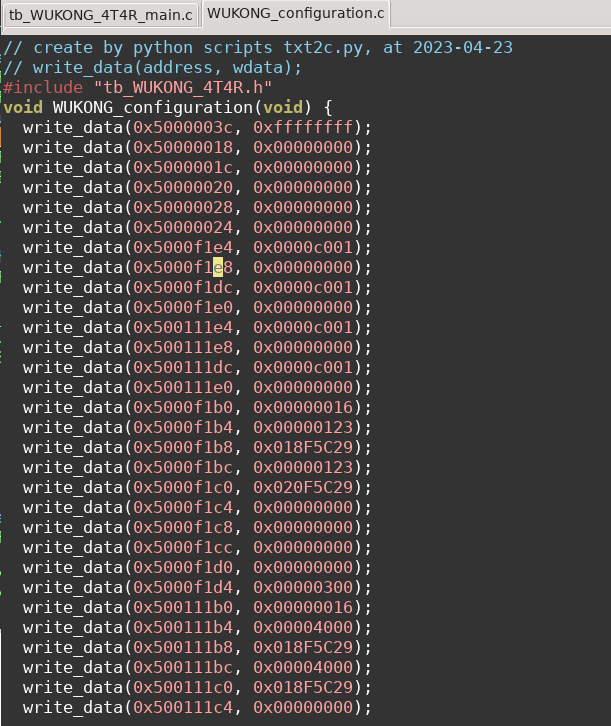
调用WUKONG\_configuration()

tb\_WUKONG\_4T4R\_main.c：



调用WUKONG\_configuration()

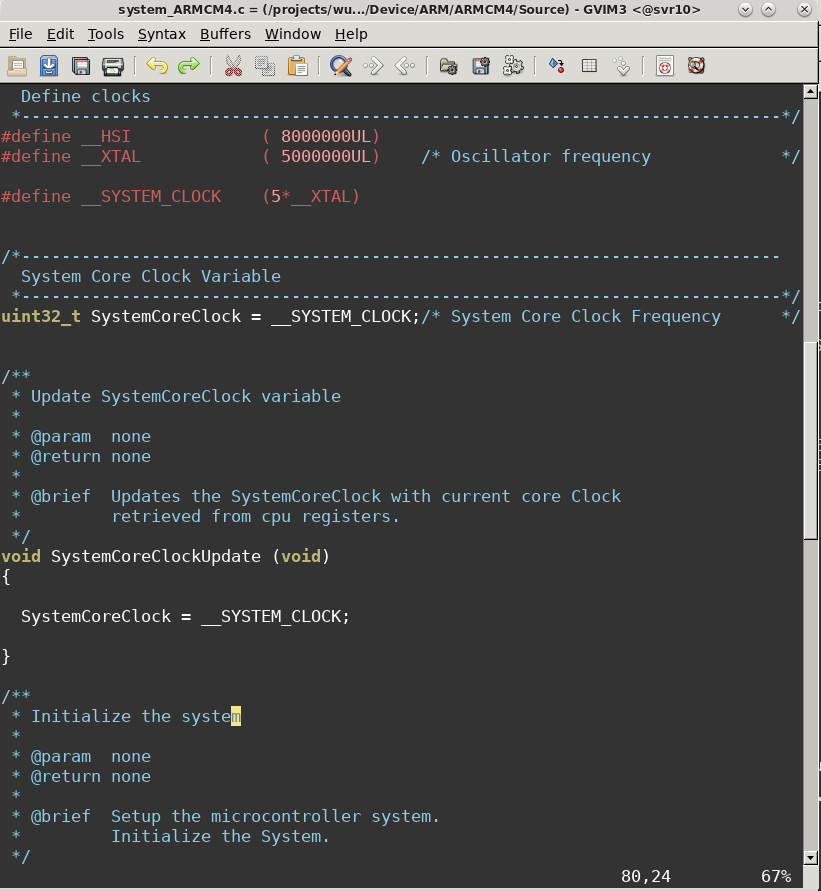
WUKONG\_configuration.c：



WUKONG\_configuration()函数主体，包含寄存器配置代码

**(c) /projects/wukong\_m4f/common/gcc\_dependence/CMSIS/4.2.0/Device/ARM/ARMCM4/Source/system\_ARMCM4.c：**

初始化系统，定义一些基本变量。



**(d) /projects/wukong\_m4f/common/gcc\_dependence/CMSIS/4.2.0/Device/ARM/ARMCM4/Source/GCC/startup\_ARMCM4\_ap\_cp.S：**

这个文件是启动文件，定义核架构，定义栈、堆的起始位置和大小，定义中断向量地址，以及定义一些启动时候的操作，如：调用**system\_ARMCM4.c**中的SystemInit函数，将ROM中的data字段复制到RAM中，清除BSS字段，定义中断函数句柄名称等。

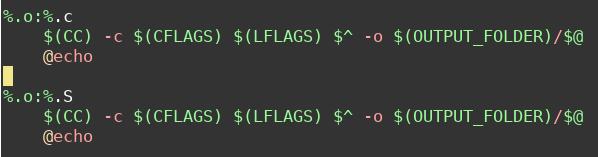
**(e) bench/ap\_cp.h：**

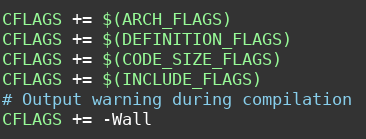
作为项目嵌入式工程的核心头文件，这个文件主要定义了跟外设相关的宏和结构体，主要包含了中断函数的中断号，各个寄存器的结构体，寄存器的偏移地址等。

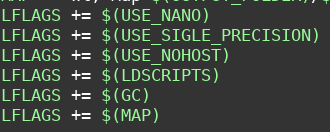
**(f) CMSIS库文件：**

Arm 芯片适配的库，与ap\_cp.h相对应，其中封装了跟cpu相关的宏和结构体。

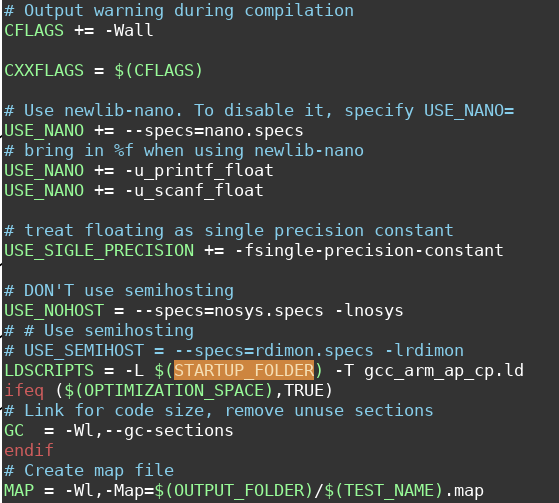
### 3.1.2脚本











## 链接

### 3.2.1文件

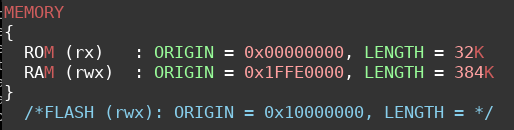
用到的文件：

* 在3.1中所用到的所有文件所生成生成的.o机器码文件，放在tb/tb\_\*/work/output中
* 链接文件

/projects/wukong\_m4f/common/gcc\_dependence/CMSIS/4.2.0/Device/ARM/ARMCM4/Source/GCC/gcc\_arm\_ap\_cp.ld（用于bootrom并执行rom中代码）

gcc\_arm\_ap\_cp.ld中都定义了memory的地址分配，各个汇编文件字段（.data和.text字段等）在最终生成的目标文件中的排列方式，定义了cpu启动时的函数入口（ResetHandler），定义了栈顶的位置等，还会检查堆栈的设置是否会溢出。

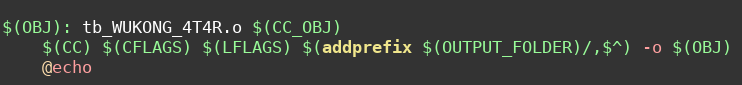
Memory的地址分配：



链接过程生成的文件：

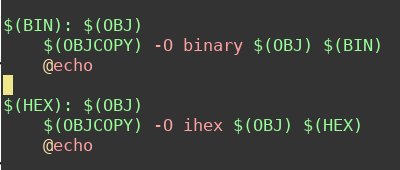
tb/tb\_\*/work/output/output.o

### 3.2.2脚本



## 格式转换

将tb/tb\_\*/work/output/output.o转换为output.hex和output.bin，bin文件作为rom的初始化文件，用于前仿，hex可用编辑器打开直接阅读。



# 反汇编

使用 make gcc\_objdump 可执行反汇编，在terminal打印反汇编信息并在work中生成objdump.log文件。

