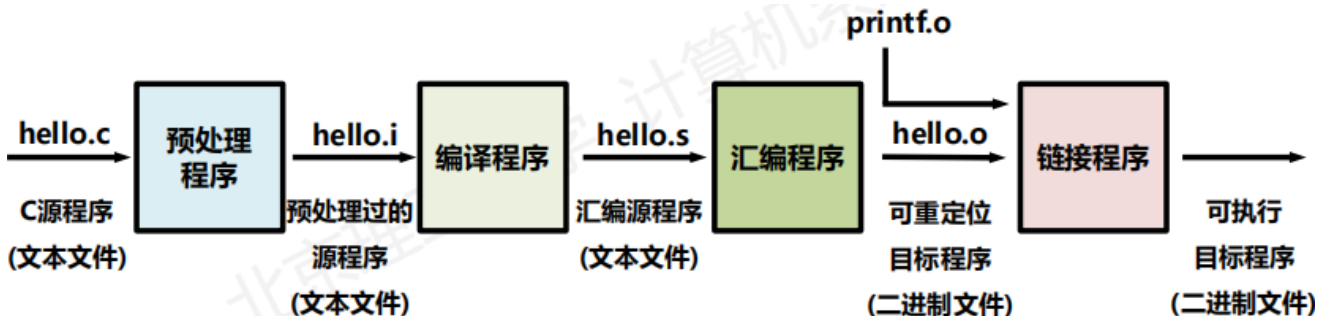


## 0512讲稿——编译原理、tcc源码

- 0512讨论安排
- 时间：5月12号晚上6:30开始，结束时间不定（内容较多，建议晚上时间“全部留出来”）
- 地点：老地方，带插排和充电器（方便使用电脑进行尝试和演示）
- 内容：知识分享（编译原理、gcc代码分析、龙芯架构文档分析）；后续流程安排（当前事务、成员分工）；补充环境和工具介绍（git、工作流等）
- 建议：
  - 由于本次聚会内容较多，因此请务必会前“保持良好精神状态”，会上定期集体休息
  - 知识分享的“形式不限”，以能够“讲明白”为准，要注意观察听者（目标是所有人讲懂）
  - 听者要“记录”必要的笔记（即便讲解者会提供资料），会上“随时提问”（目标是所有人听懂）
- 补充：
  - 翟老师新建议

### 编译原理



### 现场展示

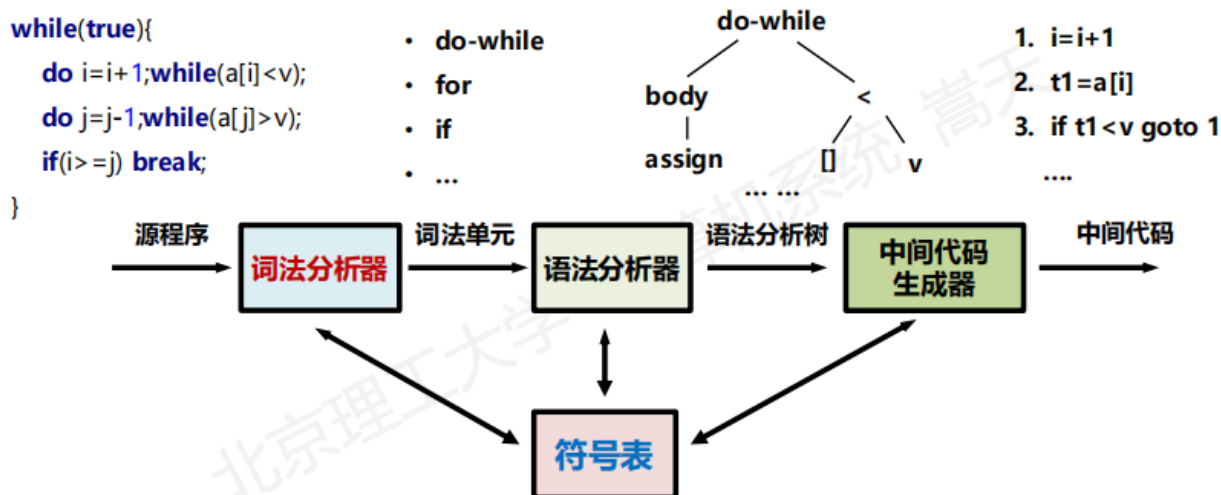
- 预处理程序：进行查找、替换、删除等文本型操作（对照tcc源文件解释）

命令	含义
#define	定义一个预处理宏
#undef	取消宏的定义
#if	编译预处理中的条件命令, 相当于C语法中的if语句
#ifdef	判断某个宏是否被定义, 若已定义, 执行随后的语句
#ifndef	与#ifdef相反, 判断某个宏是否未被定义
#elif	若条件不满足, 则执行#elif之后的语句, 相当于C语法中的else-if
#else	若条件不满足, 则执行#else之后的语句, 相当于C语法中的else
#endif	#if, #ifdef, #ifndef这些条件命令的结束标志
#defined	与#if, #elif配合使用, 判断某个宏是否被定义

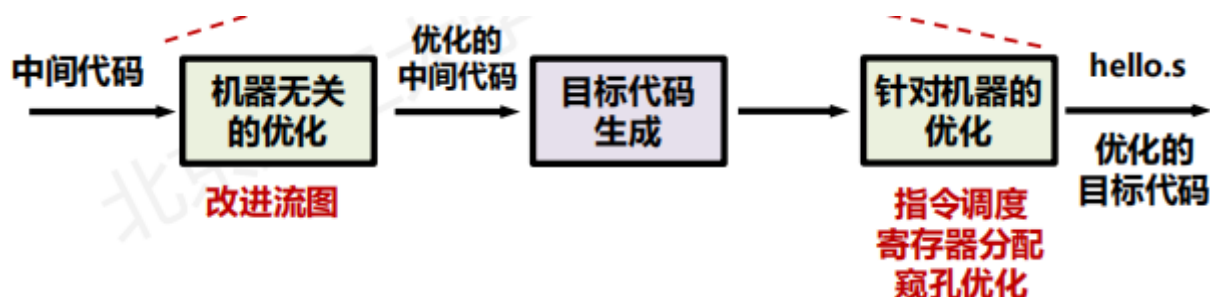
## 常用的预处理命令

命令	含义
#include	包含文件命令
#include_next	与#include相似, 但它区分<file>和“file”这两种方式的引入
#line	标志该语句所在的行号
#	将宏参数替代为以参数值为内容的字符串常量
##	将两个相邻的标记(token)连接为一个单独的标记
#pragma	说明编译器信息
#warning	显示编译警告信息
#error	显示编译错误信息

- 编译前端（分析）：词法分析、语法分析、语义分析、中间代码生成（理解代码）
  - 相关概念：符号表，中间表示
  - 和架构基本无关
  - 结果：中间代码（四元组）



- 编译后端（综合）：代码优化、目标代码生成（构建目标代码） it is an apple
  - 根据中间表示和符号表来构建目标程序
  - 和架构高度关联（是本项目的核心）



- 目标代码生成
  - 代用信息与活跃信息
    - 解决的问题：充分利用寄存器、减少冗余的内存读写操作
    - 算法（ppt）
  - 寄存器描述和变量地址描述
    - 变量地址描述数组AVALUE
    - 寄存器描述数组RVALUE
  - 代码生成算法、寄存器分配算法
    - 相关概念：四元式、变量现行值存放位置、LD/ST指令
    - 寄存器分配算法、生成存数指令
  - 示例？

- tcc特点

## 项目描述

Tiny C Compiler (TCC) 是用C语言实现的一个C语言编译器。它自身体积非常小，编译/链接速度非常快，可以自举（自己可以编译自己）。是“单趟编译器”（one-pass compiler）——它的预处理（tccpp.c）、词法分析（tccpp.c）、语法分析（tccgen.c）、类型检查（tccgen.c）、代码生成（-gen.c）、汇编（其实直接生成了机器码，不经过汇编）、链接，全部都是在一趟里完成的。目前支持x86、arm、arm64等。项目目标是将tcc移植到LoongArch平台。

- 解析我们的工作
  - 对编译器的描述：本身以A语言表达，能将B语言翻译成C语言
  - 现有：

- tcc源码(1) (能将C语言代码翻译成x64平台机器语言)
- x64平台gcc/tcc编译器(2)
- loongarch平台gcc编译器(3)
- 目标: 修改的tcc源码(4) (能将C语言代码翻译成x64/loongarch64平台机器语言)
- 路径:
  - 路径1:
    - 修改(1), 得到(4)
    - 将(4)交给(2)编译, 得到在x64平台运行、但能将C语言代码翻译成loongarch64机器语言的tcc编译器 (记为(5))
    - 将(4)交给(5)编译, 得到在loongarch平台运行、能将C语言代码翻译成loongarch64机器语言的tcc编译器
  - 路径2: (简单)
    - 修改(1), 得到(4)
    - 将(4)交给(3)编译, 直接得到在loongarch平台运行的, 能将C语言代码翻译成loongarch64机器语言的tcc编译器

## TCC源码解读

### 预备知识

## tcc初体验

现场使用展示 (x86平台编译安装、基本功能、帮助信息、选项)  
代码展示 (函数、声明、预处理命令等)

## 总体介绍

- 上下文变量s(TCCState\*), argv, ppfp
  - ppfp指向输出文件流, 会作为s的参数s→ppfp

和架构有关的声明

```

/* default target is I386 */
#if !defined(TCC_TARGET_I386) && !defined(TCC_TARGET_ARM) && \
    !defined(TCC_TARGET_ARM64) && !defined(TCC_TARGET_C67) && \
    !defined(TCC_TARGET_X86_64)
# if defined __x86_64__ || defined _AMD64_
#  define TCC_TARGET_X86_64
# elif defined __arm__
#  define TCC_TARGET_ARM
#  define TCC_ARM_EABI
#  define TCC_ARM_HARDFLOAT
# elif defined __aarch64__
#  define TCC_TARGET_ARM64
# else
#  define TCC_TARGET_I386
# endif
# ifdef _WIN32
#  define TCC_TARGET_PE 1
# endif
#endif

```

在TCCState中，描述了：

- 大量的标记和控制变量，表达配置信息和执行状态
- 维护中间信息和输出信息

```

addr_t text_addr; /* address of text section */
int has_text_addr;

/* sections */
Section **sections;
int nb_sections; /* number of sections, including first
dummy section */

```

- 其他中间信息？还没找到

阅读使用的手段

- 从TCCState的描述出发，查看标记变量的可能数值的含义

C

```
/* output format, see TCC_OUTPUT_FORMAT_xxx */  
int output_format;
```

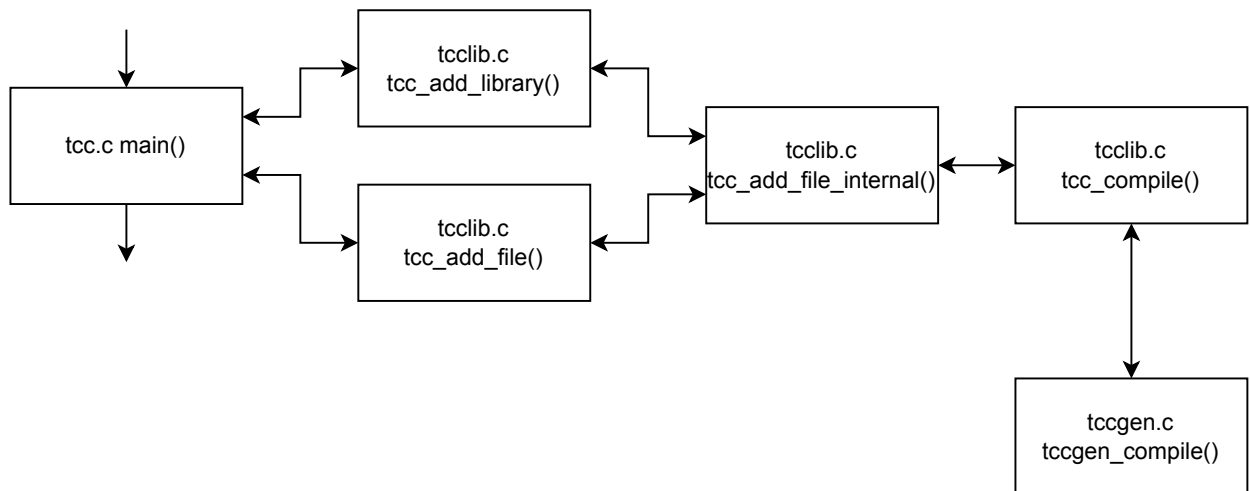
- 从TCCState的描述出发，搜索相应变量出现的位置和行为

C

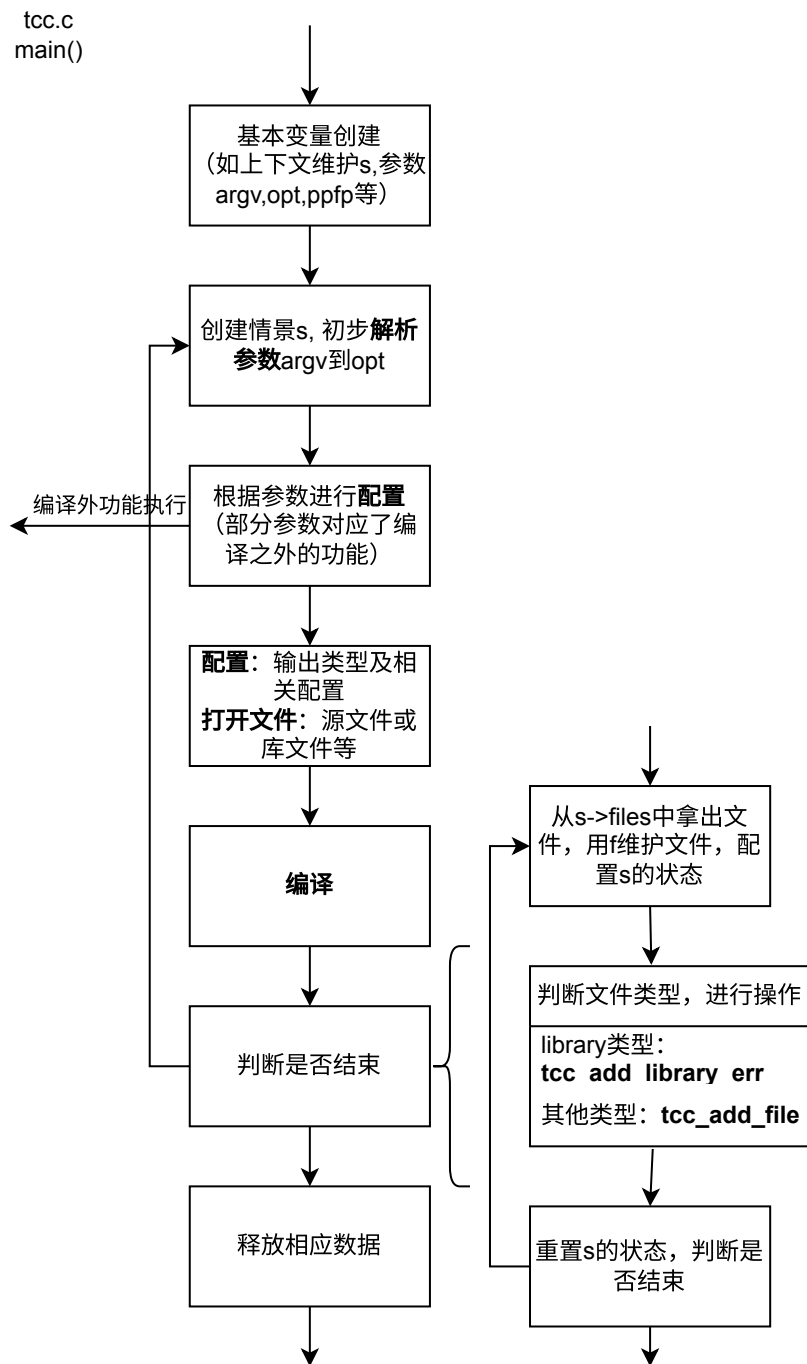
```
addr_t text_addr; /* address of text section */  
int has_text_addr;  
...  
/* library paths */  
char **library_paths;  
int nb_library_paths;
```

## 从流程出发

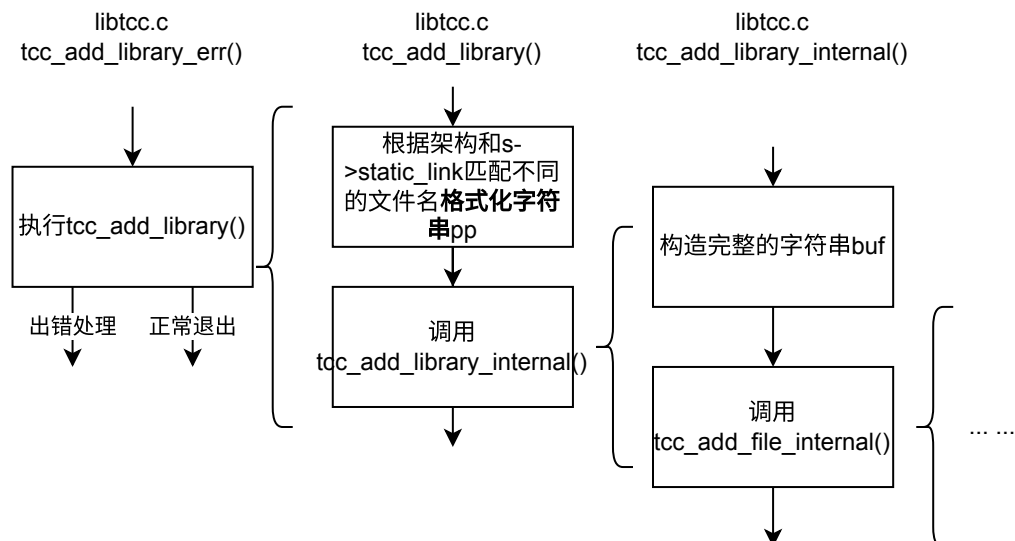
- 总体结构



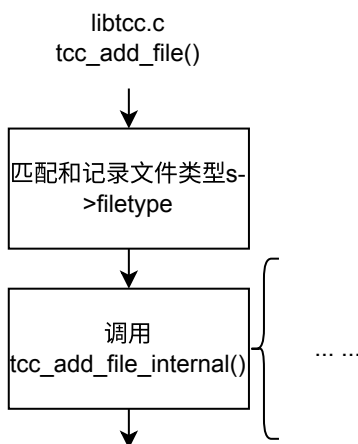
- 主函数tcc.c main()：解析参数、配置、调用库文件添加和源文件添加函数



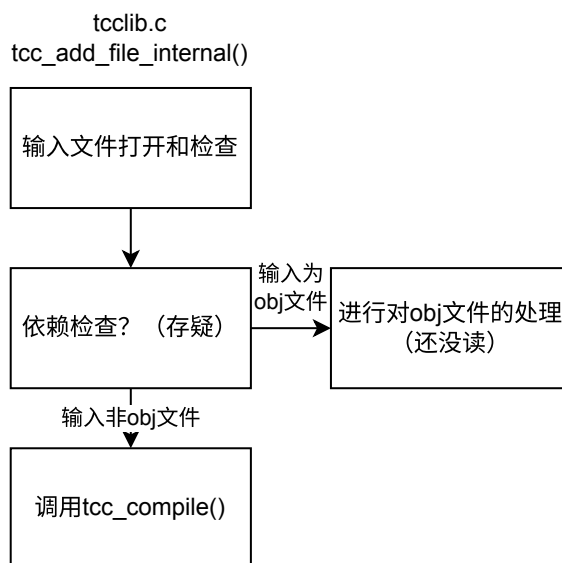
- 库文件添加libtcc.c tcc\_add\_library\_err()及相关函数：构建文件名的格式化字符串、调用添加文件的实现函数



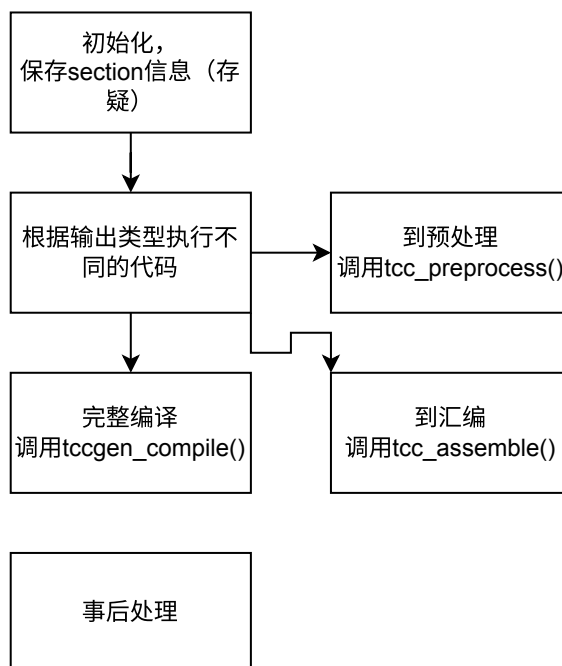
- 普通文件添加libtcc.c tcc\_add\_file()及相关函数：匹配文件类型、调用添加文件的实现函数



- tcc\_add\_file\_internal(): 打开文件、检查、处理obj文件、调用编译函数



- 核心编译代码tcc\_compile()及相关文件：编译初始化、调用仅预处理、调用仅汇编、调用完整编译函数、事后处理



- 核心代码tccgen.c tccgen\_compile()及相关代码  
看文件



## loongarch文档解读

- 只考虑基础指令？
- 之后再考虑机器相关的优化
- 先只考虑64bit
- 调试？
- 缺少的：tcc、文档和编译后端资料（tcc的更完整结构、核心代码）；改代码（可以分工；如只编译的路径、只预处理的路径等，从路径上寻找需要修改的部分）
  - 找资料（俩人）
  - 编译安装？
  - tcc、文档（李彦君）
  - 搞完之后，改代码