**编译原理与设计**

**实验报告**

实验名称： Lab 2：编译器认知实验

1. **实验目的和内容**

**实验目的：**本实验的目的是了解工业界常用的编译器GCC和LLVM，熟悉编译器的安装和使用过程，观察编译器工作过程中生成的中间文件的格式和内容，了解编译器的优化效果，为编译器的学习和构造奠定基础。

**实验内容：**本实验主要的内容为在Linux平台上安装和运行工业界常用的编译器GCC和LLVM，如果系统中没有安装，则需要首先安装编译器，安装完成后编写简单的测试程序，使用编译器编译，并观察中间输出结果。

具体步骤如实验指导文档所示。

1. **实验环境**

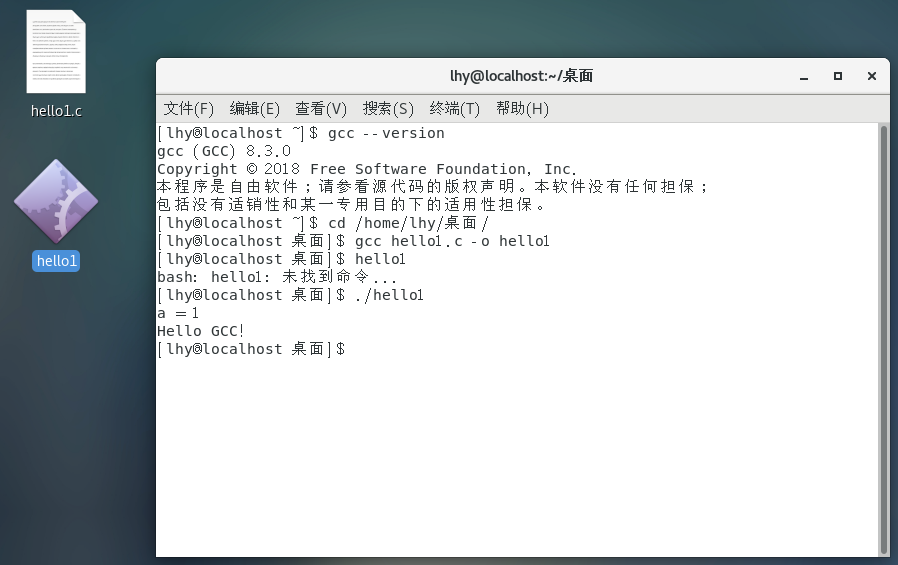
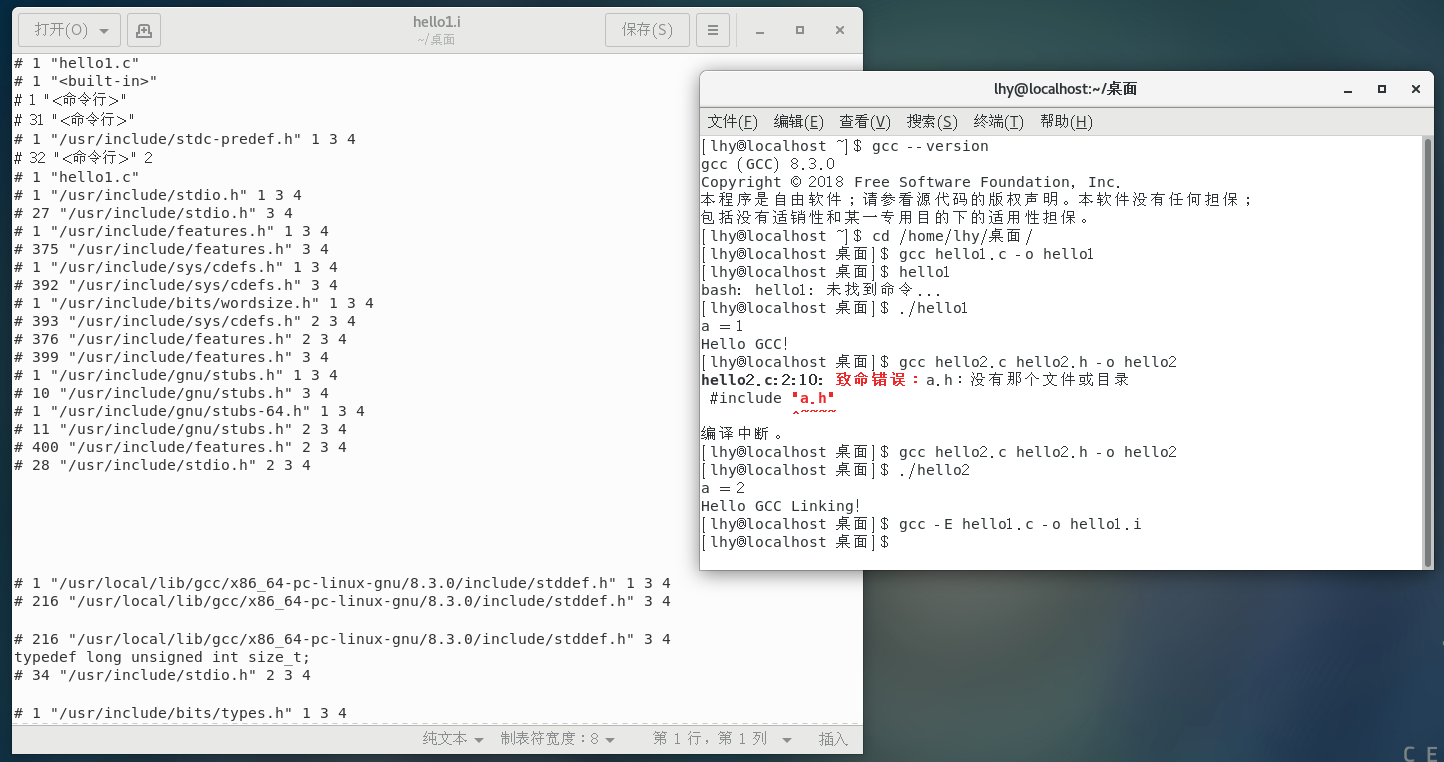
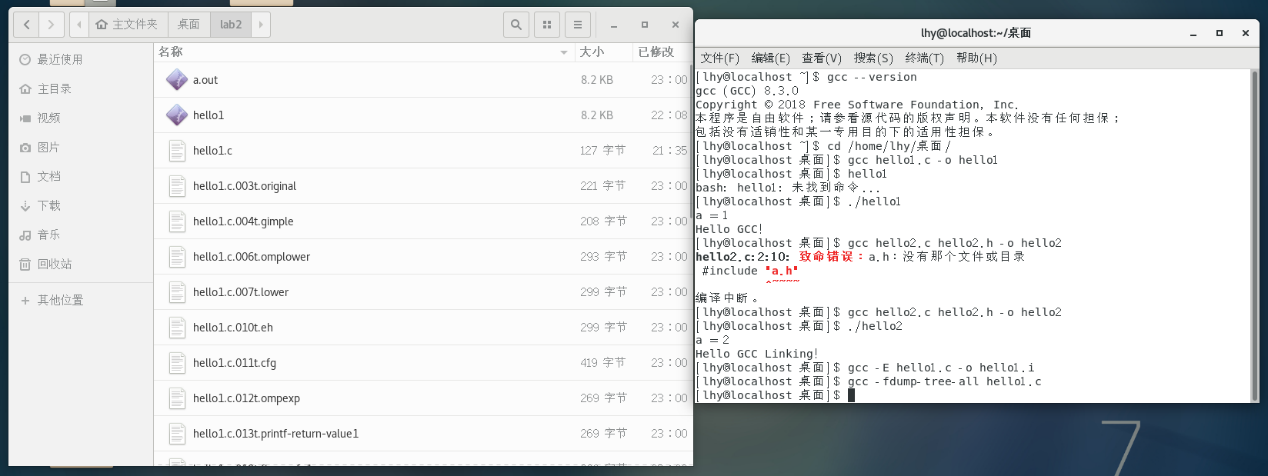
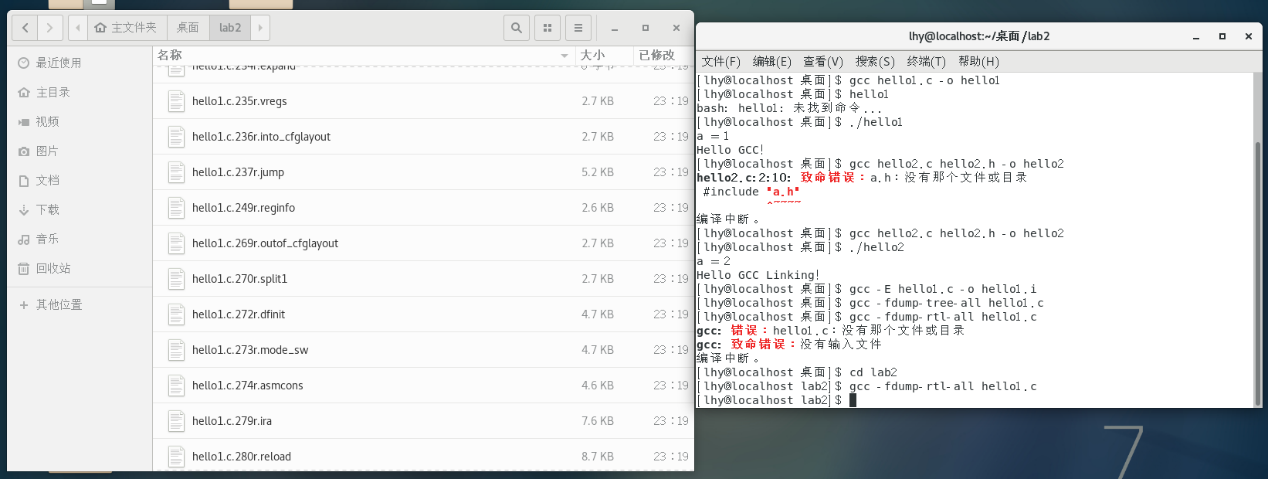
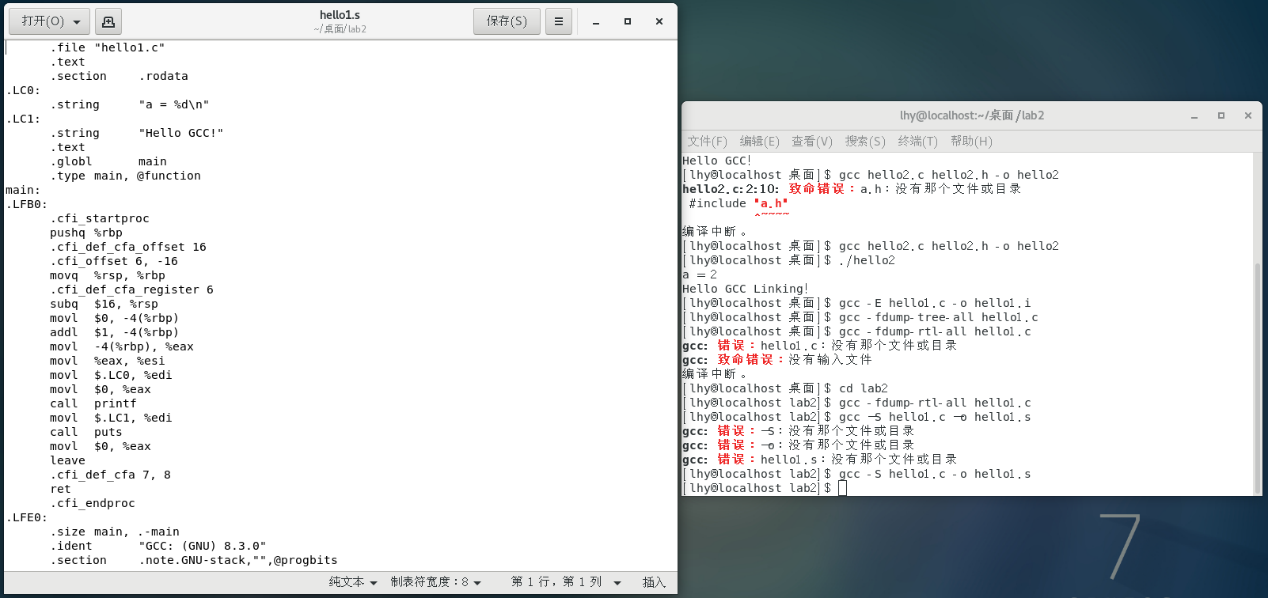
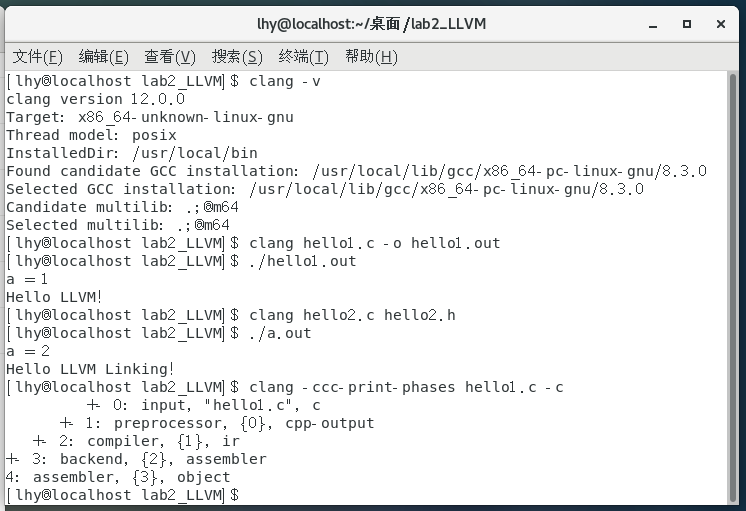
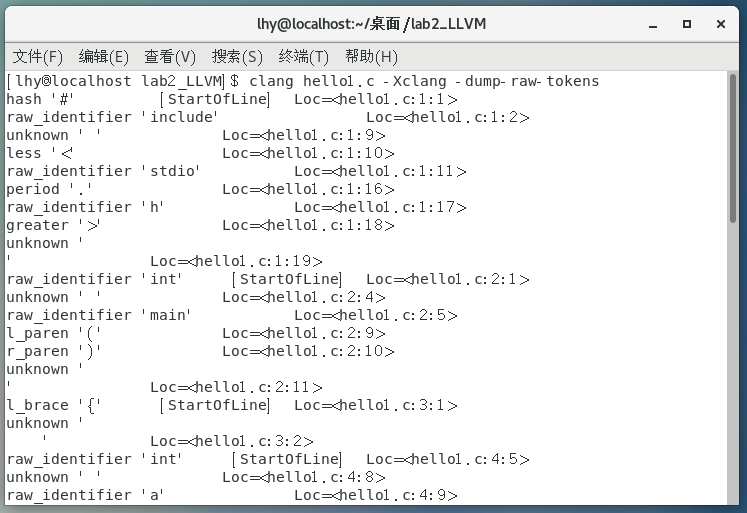
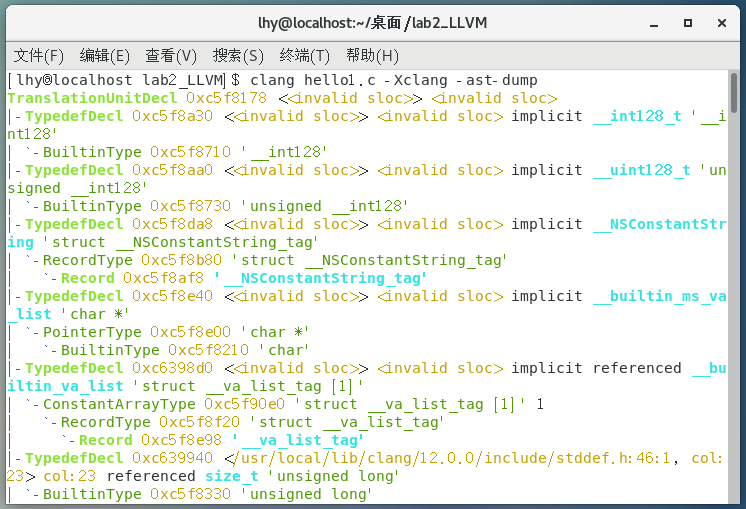
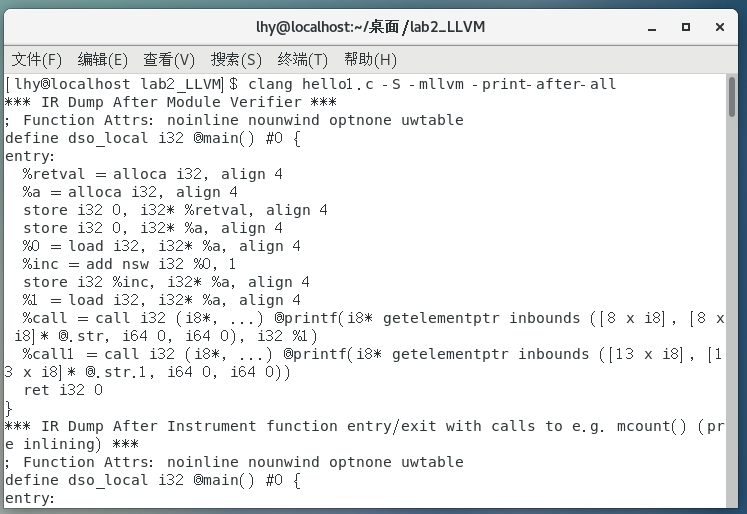
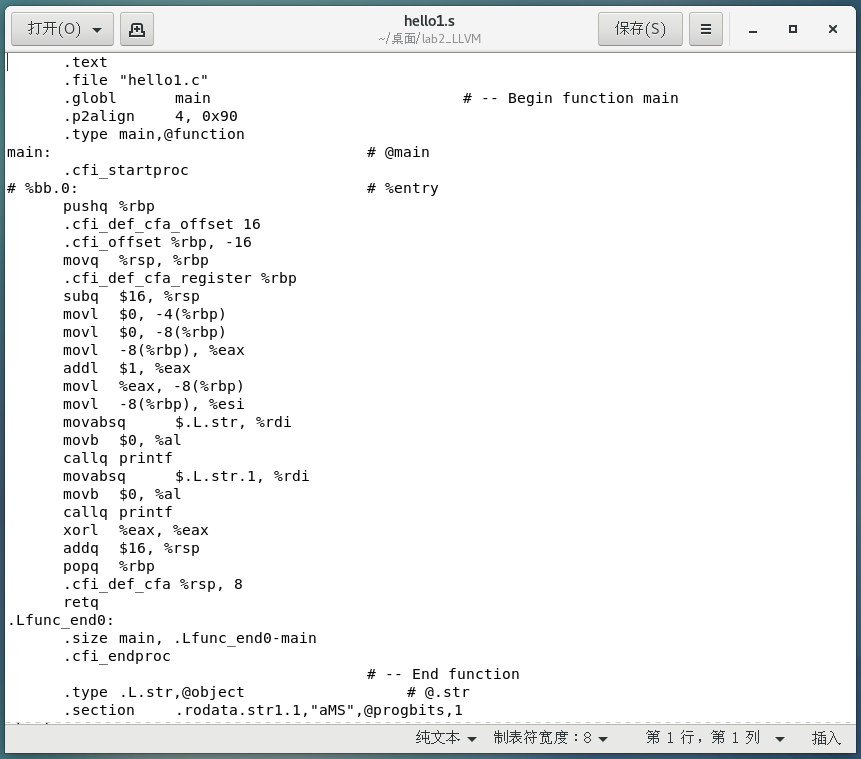
设备：RedmiBook 14 锐龙版

操作系统：Windows 10 Pro, 64-bit (Build 19045.2604) 10.0.19045

虚拟机：VMware Workstation 16 Pro (16.2.4 build-20089737)

Linux操作系统：CentOS-7.9.2009-x86\_64

GCC: 8.3.0 clang: 12.0.0

1. **实现的具体过程和步骤**
2. **GCC：**
   1. 查看编译器版本: gcc –-version
   2. 使用编译器编译单个文件: gcc hello1.c -o hello1
   3. 使用编译器编译链接多个文件: gcc hello2.c hello2.h -o hello2
   4. 查看预处理结果: gcc -E hello1.c -o hello1.i
   5. 查看语法分析树: gcc -fdump-tree-all hello1.c
   6. 查看中间代码生成结果: gcc -fdump-rtl-all hello1.c
   7. 查看生成的目标代码（汇编代码）: gcc –S hello1.c –o hello1.s
3. **LLVM：**
   1. 查看编译器的版本: clang -v
   2. 使用编译器编译单个文件: clang hello1.c -o hello1
   3. 使用编译器编译链接多个文件: clang hello2.c hello2.h
   4. 查看编译流程和阶段：clang -ccc-print-phases hello1.c -c:
   5. 查看词法分析结果：clang hello1.c -Xclang -dump-tokens:
   6. 查看词法分析结果 2：clang hello1.c -Xclang -dump-raw-tokens:
   7. 查看语义分析结果：clang hello1.c -Xclang -ast-dump:
   8. 查看语义分析结果 2：clang hello1.c -Xclang -ast-view:
   9. 查看编译优化的结果：clang hello1.c -S -mllvm -print-after-all:
   10. 查看生成的目标代码结果:
4. **GCC 运行结果分析**

将Lab1的矩阵相乘算法稍加修改后，作为我们编译器运行结果的测试程序。每次相乘计算运行时间，取五次运行时间的平均值作为结果。

* 1. gcc test.c -O0 -o test.o

Time-0:4477\*10^-6s; Time-1:4088\*10^-6s; Time-2:4415\*10^-6s; Time-3:4498\*10^-6s; Time-4:3301\*10^-6s.

Average time: 4155\*10^-6s.

* 1. gcc test.c -O1 -o test.o

Time-0:811\*10^-6s;Time-1:679\*10^-6s;Time-2:605\*10^-6s;Time-3:658\*10^-6s;Time-4:693\*10^-6s.

Average time: 689\*10^-6s.

* 1. gcc test.c -O2 -o test.o

Time-0:1066\*10^-6s;Time-1:981\*10^-6s;Time-2:1012\*10^-6s;Time-3:964\*10^-6s;Time-4:979\*10^-6s.

Average time: 1000\*10^-6s.

* 1. gcc test.c -O3 -o test.o

Time-0:315\*10^-6s;Time-1:390\*10^-6s;Time-2:260\*10^-6s;Time-3:260\*10^-6s;Time-4:261\*10^-6s.

Average time: 297\*10^-6s.

1. **LLVM 运行结果分析**
   1. clang test.c -O0 -o test.o

Time-0:5972\*10^-6s;Time-1:4315\*10^-6s;Time-2:4038\*10^-6s;Time-3:4150\*10^-6s;Time-4:6364\*10^-6s.

Average time: 4967\*10^-6s.

* 1. clang test.c -O1 -o test.o

Time-0:874\*10^-6s;Time-1:837\*10^-6s;Time-2:875\*10^-6s;Time-3:869\*10^-6s;Time-4:649\*10^-6s.

Average time: 820\*10^-6s.

* 1. clang test.c -O2 -o test.o

Time-0:312\*10^-6s;Time-1:385\*10^-6s;Time-2:258\*10^-6s;Time-3:260\*10^-6s;Time-4:354\*10^-6s.

Average time: 313\*10^-6s.

* 1. clang test.c -O3 -o test.o

Time-0:294\*10^-6s;Time-1:297\*10^-6s;Time-2:360\*10^-6s;Time-3:244\*10^-6s;Time-4:242\*10^-6s.

Average time: 287\*10^-6s.

1. **GCC 与 LLVM 对比分析**

从运行时间对比分析，可以看出GCC在多个性能优化方面都较于LLVM更有优势。但在O2或O3的某些方面，LLVM有着更高的性能提升。因此GCC与LLVM各有优势，编译器的选择也应取决于特定的应用场景。

1. **实验心得体会**

本次实验是编译原理与设计的第二次实验，在Linux环境下，使用GCC，并首次安装和使用LLVM，熟悉编译器的安装和使用过程，更通过实验和数据了解到了不同编译器的优化效果，为后续编译器进一步的学习和设计奠定了坚实的基础。

需要强调的是，由于使用的Linux操作系统CentOS7过于老旧，GCC与LLVM版本较低，因此本次实验使用了更高版本的GCC(8.3.0)与LLVM（12.0.0）。由于LLVM需求更高版本的CMake、Python、clang等，本人选择下载源码后进行编译安装，进而耗费了近一天半的时间，实属进度缓慢、耗时过多。必须汲取经验，提高效率。