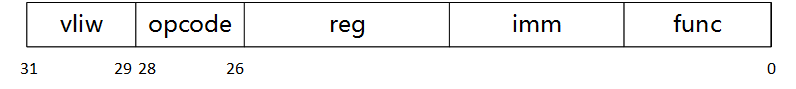
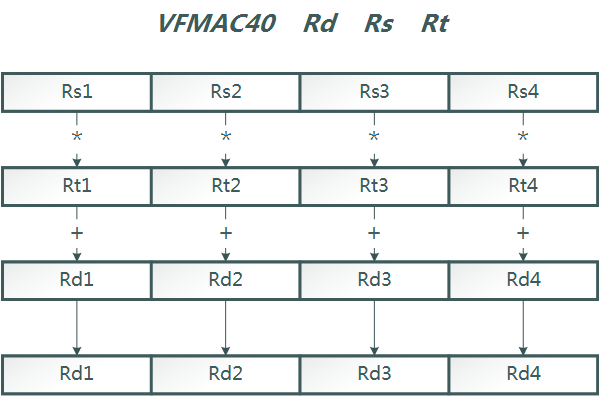
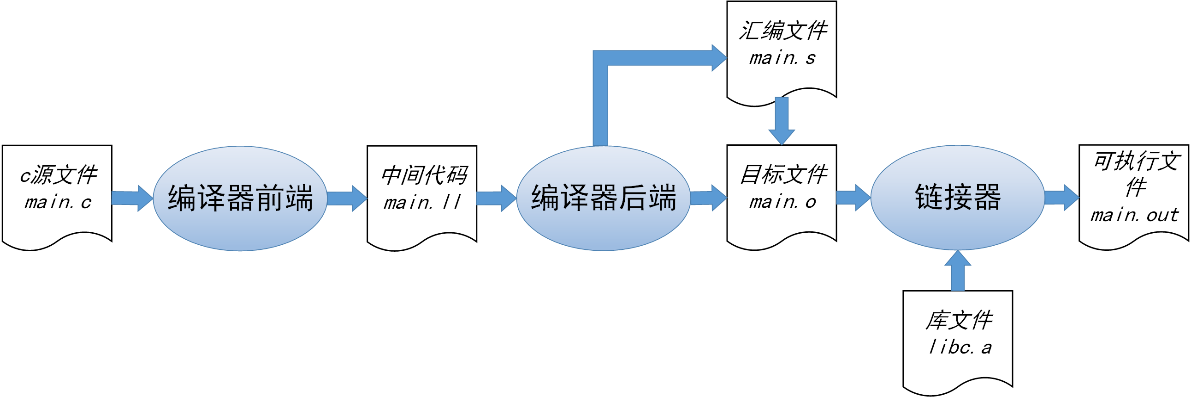
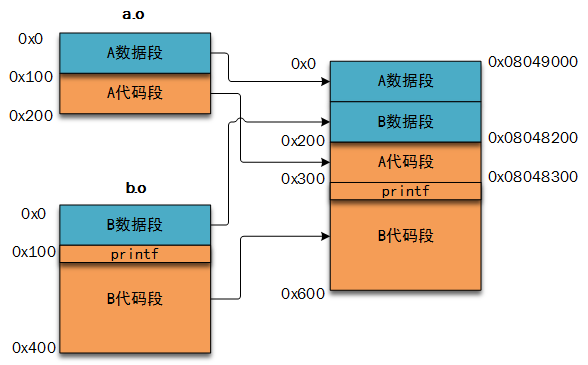
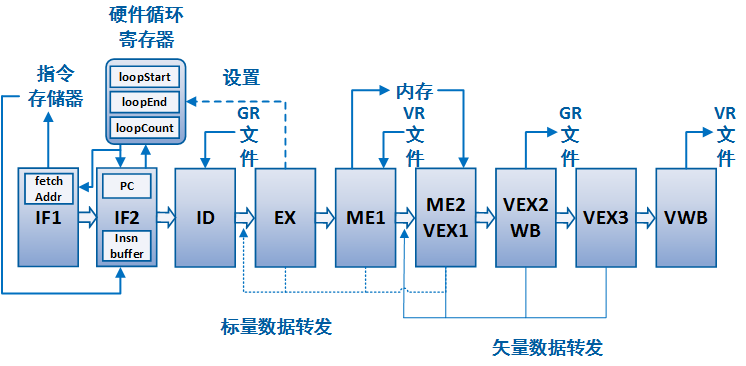
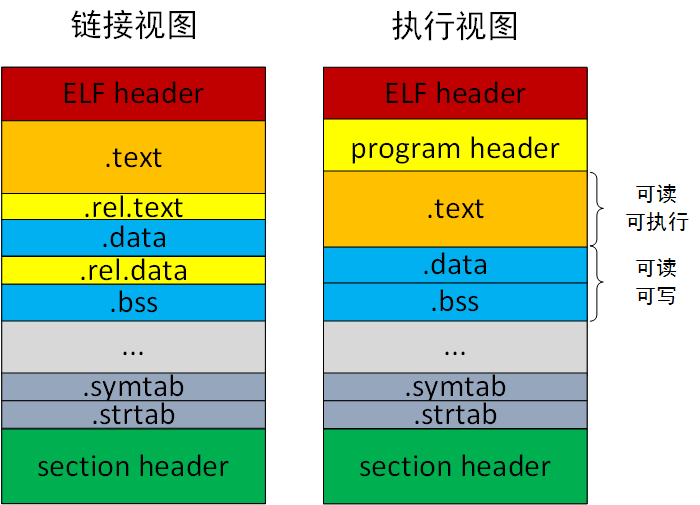
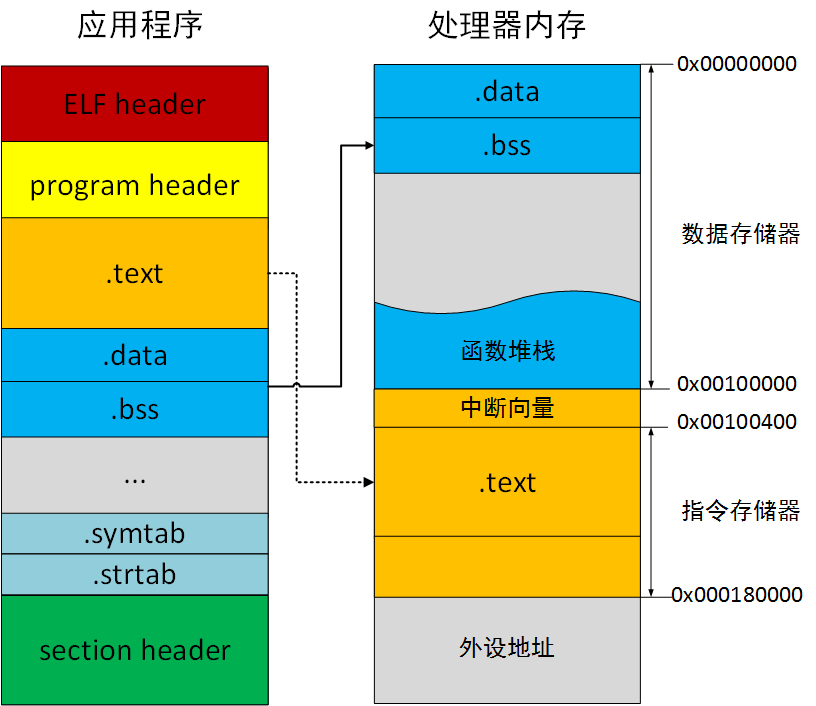
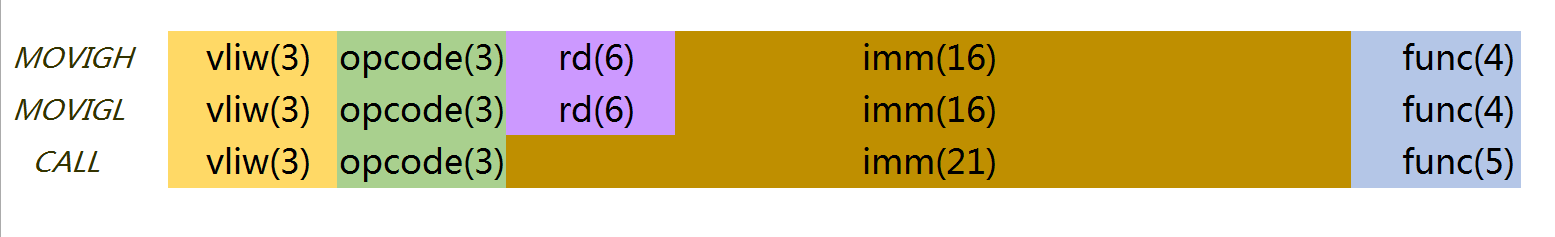
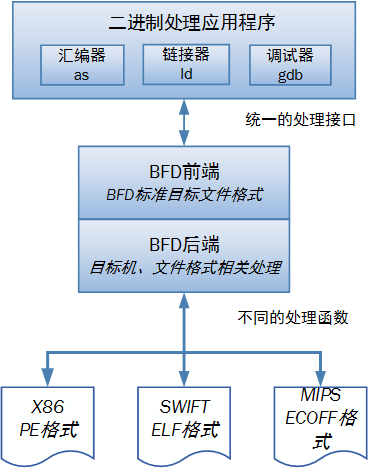
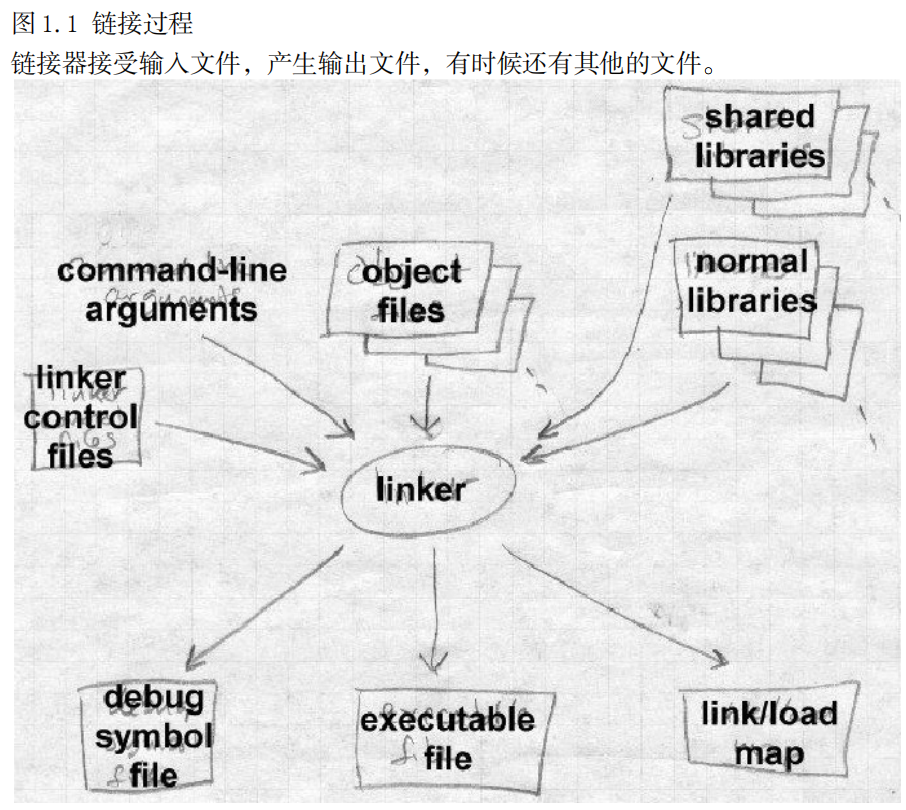
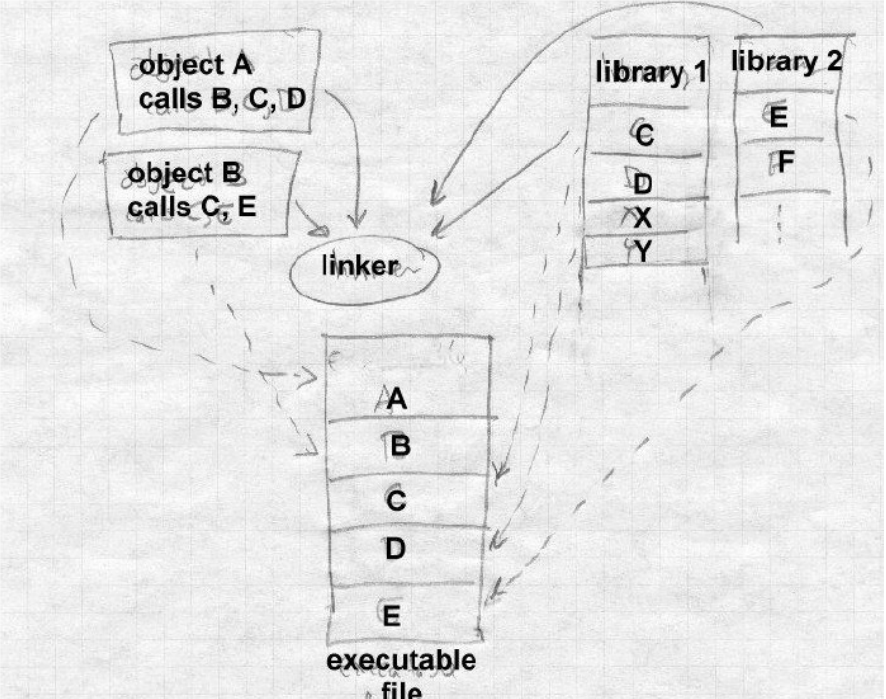
1. <https://sourceware.org/binutils/>
2. GNU toolchain（GNU工具链）：
   1. GNU make：用于编译和构建的自动工具；
   2. GNU编译器集合（GCC）：一组多种编程语言的编译器；
   3. GNU Binutils：汇编链接器https://sourceware.org/binutils/binutils-porting-guide.txt
      1. ld ：链接器
      2. gas：汇编器目标特定的汇编程序代码保存在config子目录中。
      3. gold ：新链接器，目前还在开发中
      4. gprof ：显示配置信息，没有目标代码
      5. opcodes：汇编/反汇编库
      6. include:主要头文件
      7. BFD：二进制文件描述符库，处理ELF、COFF、SREC等。如果必须识别新的对象文件类型，那么应该在这里添加支持它的代码。
      8. cpu:CGEN?为opcodes或GDB中的SIM模拟器自动生成特定源文件
      9. binutils：不是主目录，而是binutils各个工具（objcopy, objdump and readelf等）的顶级源目录
         1. objcopy：目标文件格式转换工具
         2. objdump：目标文件信息打印工具
         3. readelf：ELF格式文件分析工具
      10. elfcpp：只为gold读写ELF服务
      11. intl：GNU gettext库
      12. libiberty：在POSIX和glibc之前，这是一个GNU项目，提供一组标准函数。标准的功能。它仍然存在于binutils。最有价值的是它的免费存储管理和参数解析功能。
   4. GNU Debugger（GDB）：代码调试工具；
3. 链接器（Linker）：
   1. 传统：对编译生成的一个或多个目标文件进行合并和符号链接，生成可执行文件或共享库。
      1. 地址与空间分配：第一遍扫描
      2. 符号解析和重定位：第二遍扫描
   2. 现在：链接器对每一个程序的部分地址进行绑定并分配相对地址：加载器（Loader）完成最后的重定位步骤并赋予的实际地址
   3. **程序加载**：数据/代码/程序拷贝入内存
   4. **重定位**（两遍链接）：对于链接器的一种普遍情景是由多个子程序来构建一个程序，并生成一个链接好的起始地址为 0 的输出程序，各个子程序通过重定位在大程序中确定位置。当这个程序被加载时，系统会选择一个加载地址，而链接好的程序会作为整体被重定位到加载地址。
   5. **符号解析**：当通过多个子程序来构建一个程序时，子程序间的相互引用是通过符号进行的；主程序可能会调用一个名为 sqrt 的计算平方根例程，并且数学库中定义了sqrt 例程。链接器通过标明分配给 sqrt 的地址在库中来解析这个符号，并通过修改目标代码使得 call 指令引用该地址
4. ELF：可执行链接格式是 UNIX 系统实验室开发的目标文件格式，ELF文件是以多个不同属性的段作为文件的基本构成单位。编译生成的目标文件格式；程序的可执行文件；动态库的共享目标文件；用于调试的核心转储文件；
   1. ELF头（ELF Header）：全局的描述信息，记录整个文件的基本属性。
   2. 段头表（Section Header Table）：ELF文件中各个段的索引和基本属性。
   3. 代码段：以包含程序主体指令的.text段为主。
   4. 数据段：存储的都是静态类的变量，典型的有已经初始化的全局或局部静态变量（.data），未初始化的全局或静态变量（.bss），字面值常量或const变量（.rodata）等。
   5. 符号表（.symtab）：符号表中记录了这个目标文件所用到的所有符号信息，包括符号名，符号值，符号大小，符号类型和绑定信息以及符号定义所在的段。
   6. 重定位表：既有对代码段的重定位表（.rel.text），也有对数据段的重定位表（.rel.data）。表中每一项表示程序中一个需要解析和重定位的符号，它记录有重定位要修正的地址，修正的方式和所关联符号的索引。重定位过程就是通过重定位表和符号表的关联实现的。
5. SWITF链接器重地位
   1. 代码段的重定位
      1. MOVIGH（R\_SWIFT\_HI16）：立即数字段保存静态类型变量地址的高16位
      2. MOVIGL（R\_SWIFT\_LO16）：立即数字段保存静态类型变量地址的低16位
      3. CALL（R\_SWIFT\_PC21）：立即数字段改为函数定义点相对于调用点PC的偏移
      4. 指针引用（R\_ SWIFT\_32）：static int a; static int \*pa = &a;
   2. 数据段的重定位
6. 编译器驱动：UNIX make 程序就是典型的例子，仅编译那些被修改过的源文件
   1. C 语言预处理器处理 A，生成预处理的 A
   2.  C 语言编译预处理的 A，生成汇编文件 A
   3.  汇编器处理汇编文件 A，生成目标文件 A
   4.  C 语言预处理器处理 B，生成预处理的 B
   5.  C 语言编译预处理的 B，生成汇编文件 B
   6.  汇编器处理汇编文件 B，生成目标文件 B
   7.  链接器将目标文件 A、B 和系统 C 库链接在一起
7. 链接器命令语言：控制链接过程
   1. 命令行: 多数系统都会有命令行（或相似功能的其它程序），通过它可以输入各种文件名和开关选项。这对于 UNIX 和 Windows 链接器是很常用的方法。对于那些命令行长度有限制的系统，常用的办法是让链接器从文件中读取命令并在命令行上那样对待他们。
   2.  与目标文件混在一起: 有些链接器，如 IBM 主机系统的链接器，从一个单个输入文件中接受替换的目标文件及链接器命令。这种方式来源于卡片输入的年代，那时程序员需要把目标代码卡片摞起来和手工打制的命令卡片一起送到读卡器中。
   3.  嵌入在目标文件中: 有一些目标代码格式，特别是微软的，允许将链接器命令嵌入到目标文件中。这就允许编译器将链接一个目标文件时所需要的任何选项通过文件自身来传递。例如 C 编译器将搜索标准 C 库的命令嵌入到文件中（来传递给链接过程）。
   4.  单独的配置语言: 极少有链接器拥有完备的配置语言来控制链接过程。可以处理众多目标文件类型、机器体系架构和地址空间规定的 GNU 链接器，拥有可以让程序员指定段链接顺序、合并相近段规则、段地址和大量其它选项的一套复杂的控制语言。其它链接器一般拥有诸如支持程序员可定义的重叠技术等特性的稍简单一些的配置语言
   5. 链接器可以通过设置链接脚本来控制目标文件段的地址。通过ld –T Script.lds这样的参数选项
8. 汇编语言
   1. ebp栈底高地址，esp栈顶低地址
   2. push其他一般寄存器意思是先保存之前存放在这寄存器的值，然后再拿来操作
   3. [ebp-4/8/12]局部变量，[ebp+8/12/16]传过来的参数，[ebp+4]返回地址=调用本身正在执行的程序的call指令的下一条指令地址
   4. 分成函数调用（看栈怎么变化）和代码变量操作（看寄存器里面的值变化）
9. 链接器注意点
   1. 内存地址
   2. 字节顺序和对齐
   3. 地址构成，寻址方式
   4. 指令格式
10. BSD：类UNIX操作系统中的一个分支的总称
11. GNU：基于 Linux 内核的类 Unix 操作系统
12. LLVM：和GCC同类，一种编译器，各有优缺点
13. 硬件体系结构的两个方面影响到链接器：程序寻址和指令格式
14. 应用程序二进制接口（Application Binary Interface）：包含了应用程序在这个系统下运行时必须遵守的编程约定
15. Pentium（奔腾）：英特尔第五代x86架构的微处理器
16. 重链接：一次链接器运行的输出文件可以作为下次链接器运行的输入。
17. 链接库：在某些系统上你可以直接将一些目标代码文件链接在一起
18. 目标代码库：当链接器处理完所有规则的输入文件后，如果还存在未解析的导入名称（imported name），它就会查找一个或多个库，然后将输出这些未解析名字的库中的任何文件链接进来。
19. DLL(Dynamiclly Linked Libraries):微软 Windows 动态链接库
20. x86 使用从右向左的字节序，逆序，a 定义的位置 0x1234，最终代码为34 12
21. 每一个全局符号（变量/函数名）都会以下划线做为前缀\_main
22. ./configure -build,-host,-target
    1. build:执行代码编译的主机，正常的话就是你的主机系统。这个参数一般由config.guess来猜就可以。当然自己指定也可以。
    2. host:编译出来的二进制程序所执行的主机，因为绝大多数是如果本机编译，本机执行。所以这个值就等于build。只有交叉编译的时候（也就是本机编译，其他系统机器执行）才会build和host不同。用host指定运行主机。
    3. target:这个选项只有在建立交叉编译环境的时候用到，正常编译和交叉编译都不会用到。他用build主机上的编译器，编译一个新的编译器（binutils, gcc,gdb等），这个新的编译器将来编译出来的其他程序将运行在target指定的系统上。
23. SWIFT指令编码方式
24. SWIFT中VFMAC40指令的执行过程
25. 
26. 工具链的编译链接流程
27. 目标文件的链接过程
28. SWIFT的九级流水线
29. ELF
30. SWIFT程序装载的内存布局
31. 与重定位相关的指令格式
32. SWIFT重定位类型的地址修正方式：

S：表示重定位项所引用符号的实际定义地址。

A：表示保存在被修正位置的值，用于修正重定位计算出的地址值。对于SWFIT的指令 集，它总是为0。

P：重定位项标识的被修正位置的地址。

|  |  |
| --- | --- |
| **重定位类型** | **地址修正值** |
| R\_SWIFT\_HI16 | ((S + A) >> 16) & 0xFFFF |
| R\_SWIFT\_LO16 | (S + A) & 0xFFFF |
| R\_ SWIFT\_PC21 | ((S－４－P) >> 2 << 5) & 0x03FFFFE0, |
| R\_ SWIFT\_32 | S + A |

1. BFD的前后端结构
2. 
3. 链接器读入目标文件，再跟上很多库文件
4. 