本文件为 Francis 制作

第一章 计算机系统概述

1. 计算机的诞生史
   - ◆ 世界上第一台真正意义的电子数字计算机：ABC（Atanasoff-Berry Computer，阿塔那索夫-贝瑞计算机），1935~1939 年由美国艾奥瓦州立大学物理系副教授约翰·文森特·阿塔那索夫和克里福特·贝瑞研制成功。国际计算机界公认约翰·文森特·阿塔那索夫被称为“电子计算机之父”。
   - ◆ 世界上第一台真正实用的电子计算机：ENIAC（Electronic Numerical Integrator And Computer，电子数字积分计算机），1946 年由美国宾夕法尼亚大学莫齐利、艾克特研制。
   - ◆ 现代计算机结构思想的诞生：冯·诺依曼于 1945 年发表“关于 EDVAC 的报告草案”的全新“存储程序通用电子计算机方案”。该报告中提出的计算机结构被称为冯·诺依曼结构，标志着现代计算机结构思想的诞生。
   - ◆ 1946 年，普林斯顿高等研究院（the Institute for Advance Study at Princeton，IAS ）开始按照冯·诺依曼的设计实现“存储程序”式计算机，被称为 IAS 计算机。
   - ◆ 世界上第一台存储程序计算机：1949 年由英国剑桥大学完成的 EDSAC。
2. 简述“存储程序”工作方式的基本思想。
3. 简述冯·诺依曼结构计算机的五个基本组成部分。
4. 简述 ALU、控制器、主存储器、通用寄存器、标志寄存器、指令寄存器、程序计数器、总线的作用。CPU 由上述的哪几部分组成？CPU 送到地址线的主存地址要首先存放在哪个寄存器中？发送到或从数据线取来的信息要先存放在哪个寄存器中？
5. 图 1-1：解释冯·诺依曼结构计算机的基本组成和相关工作原理。
6. 机器指令的 0/1 序列通常被划分成哪些字段？各字段的含义是什么？
7. 一条机器指令的执行过程通常包含哪几个阶段？
8. 图 1-9：简述程序的执行过程。
9. 什么是 ISA？ISA 主要包含哪些内容？

2.存储程序的工作方式基于将程序和数据统一存储在计算机内存中的思想。在这种模式下，程序和数据都以二进制代码形式存储，计算机通过读取存储器中的指令来执行任务。存储程序的核心概念是程序控制流由指令序列决定，可以根据需要修改和执行程序，提高计算机的灵活性和可编程性。

3.冯·诺依曼结构计算机的五个基本组成部分包括：中央处理单元（CPU）、存储器、输入设备、输出设备和总线系统。CPU 负责执行指令并控制计算机的操作；存储器用于存储程序和数据；输入设备提供外部数据给计算机；输出设备用于将计算机结果展示给用户；总线则连接并协调各个部分的数据传输。

4.ALU（算术逻辑单元）执行算术和逻辑运算；控制器负责控制指令的执行顺序并协调各个部件的工作；主存储器存储程序和数据；通用寄存器用于临时存储操作数和中间结果；标志寄存器用于存储操作结果的状态信息；指令寄存器存储当前执行的指令；程序计数器存储下一条要执行的指令地址。CPU 由 ALU、控制器、寄存器、指令寄存器和程序计数器等部分组成。CPU 送到地址线的主存地址通常存放在程序计数器中；发送到或从数据线取来的信息存放在寄存器中。

5.冯·诺依曼结构计算机的基本组成包括：中央处理单元（CPU）、存储器、输入设备、输出设备和总线。基本工作原理是 CPU 通过程序计数器逐条取指令、解码并执行，计算结果可以存储到内存中或通过输出设备显示。总线用于各个部分之间的数据传输。

6.机器指令的 0/1 序列通常被划分成操作码字段、操作数字段和地址字段。操作码字段指定指令的操作类型（如加法、跳转等），操作数字段指示指令的操作数（如寄存器或内存地址），地址字段则给出数据或操作数的存储位置。操作码字段决定指令的基本功能。

7.一条机器指令的执行过程通常包含取指令、解码和执行三个阶段。取指令阶段从内存获取指令，解码阶段将指令解码成具体操作，执行阶段根据指令要求进行数据操作或地址跳转等。

8.程序的执行过程通常包括加载程序、指令取出、指令解码、操作执行和结果存储五个阶段。程序从磁盘加载到内存，CPU 按顺序取指令并解码执行，操作结果可能存储到寄存器或内存中，并输出到外部设备。

8.ISA（指令集架构）是计算机硬件与软件之间的接口，定义了计算机指令的格式、功能以及操作数的寻址方式。ISA 主要包括指令集、寻址方式、寄存器、数据类型等内容，决定了计算机的操作能力和性能。

第二章 数据的机器级表示与处理

1. 信息编码的两大要素是什么？现实世界中的各种媒体信息要怎么样才能在计算机内部进行存储、运算和传送？什么是数字化编码过程？

信息编码的两大要素是**数据的表示方法**和**传输的编码规则**。现实世界中的媒体信息通过**采样、量化和编码**过程转化为数字信号，才能在计算机内存储、运算与传送。数字化编码过程是将连续信号转化为离散的数字形式，确保信息能以计算机可处理的格式存储与传输。

2. 了解什么是数值数据和非数值数据？

数值数据是表示量化或测量结果的数据，如整数和浮点数；非数值数据指无法直接量化的数据，如字符、图像、声音等

3. 数值数据表示的三要素是什么？

符号位、数值部分（有效数字）和**指数部分**

4. R 进制数和十进制数之间的相互转换。

Easy。

5. 什么是定点数？定点数有哪两种？

定点数是小数点位置固定的数值表示方式，分为**有符号定点数**和**无符号定点数**

6. 什么是真值、机器数、原码、补码、反码、移码？

真值是表示数字的实际值；机器数是其计算机表示形式；原码是直接表示数字的二进制形式；补码是表示负数的一种编码方式；反码是对原码的补充表示；移码是用偏移量调整的二进制表示

7. 原码、补码的相互转换：给出一个 n 位整数，能够正确计算它的原码和补码。

原码与补码的相互转换方法：对于负数，原码转换为补码时先取反再加 1；补码转原码则先

减 1 再取反。

8. 以时钟为例说明为什么在模运算系统中，减去一个数等于加上（这个）负数的补码？

在模运算系统中，减去一个数等于加上该数的负数补码，因为减法操作通过补码表示等价于加法操作。

9. 为什么说计算机内部的运算电路是一个模运算系统？

计算机内部运算电路采用模运算系统，因为计算机的所有运算（包括加法、减法、乘法、除法）都在有限的数值范围内通过模算术完成

10. 对给定的 C 语言运算表达式，正确计算结果并解释原因：例 2.21 和习题 34、35。

11. 浮点数的表示

（1）为什么要进行尾数的规格化？为什么规格化后尾数部分可以表示多一位的精度：用 23 个数位表示 24 位尾数？

规格化使得尾数位于一个标准范围内，增加精度。规格化后尾数部分可以表示多一位的精度，主要是因为浮点数的规格化表示中，尾数的第一位（隐含的 1）并不存储。通过这一方式，尾数部分的存储位数相对较少，但可以表示更高精度的数值。

（2）浮点数的编码需要对哪几个部分进行编码？

（3）什么是移码、偏置常数？阶码的移码和真值之间的换算关系是什么？

（4）IEEE754 浮点数编码标准：

（a）IEEE754 浮点数的尾数规格化形式是什么样的？

（b）单/双精度浮点数的尾数、阶码的位数、偏置常数各是多少？

（c）给出一个用 IEEE754 标准表示的浮点数，可以换算其十进制真值，反之亦然。

- 非规格化数：作用、特征是什么？默认阶码是多少？

允许表示一个很接近 0 的数字，特征是有一个偏移量，默认是-127

- +∞/-∞：特征是什么？解释：5/0 和 5.0/0 的区别。

符号位一个是 0 一个是 1，阶数全都是 1，尾数全都是 0

- 非数：什么是非数？非数有什么用处？了解静止的 NaN 和通知的 NaN 的特征格式什么？

尾数字段至少有一个 0，阶数字段全部是 1，符号可以是 0 或者 1

12. 了解 BCD 码、中文编码（区位码、国标码、机内码）。

BCD 码用于表示数字，每个十进制数位用 4 位二进制表示。中文编码包括区位码、国标码和机内码，用于表示汉字字符。

13. 什么是数据的宽度？

数据的宽度是指计算机中数据表示的位数，通常指处理器一次可以处理的数据量

14. 什么是数据通路？计算机系统中字长是指什么？字和字长有什么区别？

数据通路是指计算机内部用于传输数据的硬件路径，字长是计算机中数据处理的单位长度，字指的是一次运算处理的数据单位

15. 什么是最高有效字节（MSB）和最低有效字节（LSB）？

最高有效字节（MSB）是数据中最高位对应的字节，最低有效字节（LSB）是数据中最低位对应的字节

16. 什么是大端方式、小端方式？对于一个已知数据的字节数据，能够分别写出它在大端方式和小端方式下的字节排列。C 语言中数据的地址指的是 MSB 的地址还是 LSB 的地址？存放方式不同的机器间程序移植或数据通信会存在什么问题？

大端方式和小端方式指的是数据存储的字节顺序，大端将高位字节存放在低地址，小端则相反。不同的存放方式可能会影响跨平台的数据传输与程序移植。通常指的是 LSB。

17. 逻辑左移的溢出判定条件是什么？算术左移的溢出判定条件是什么？

逻辑左移溢出判断是移出的最高位。算术左移溢出则需检查符号位的变化

18. 了解 C 语言的基本运算与机器级运算之间的对应关系。

汇编。

19. 基于 n 位整数加减运算器（原理图）简述 n 位整数加减的原理：

（1）如何在同一个电路上实现加、减两种运算？

- sub 输入端的作用
- result、Cout 的输出
- ZF、CF、SF、OF 的设置

（2）一般了解反向器、多路选择器的作用

20. 整数加减：结果必须在可表示范围内，超出范围的需要加 2^n 或减 2^n。

21. 整数的乘运算：操作数长度为 n，乘积长度为 2n，数据截断

（1）原码乘法和补码直接相乘

（1）符号数乘法溢出的判断

（2）无符号数乘法溢出的判断

22. 整数的除运算：n 位整数除以 n 位整数，结果还是整数

（1）不能整除时需要进行舍入。舍入规则：朝 0 方向舍入。

（2）利用右移实现除 2^k：

不能整除时的舍入处理：低位截断、朝零舍入

- 无符号数、带符号正整数：移出的低位直接丢弃
- 带符号负整数：先加偏移量(2^k-1)，然后再右移 k 位。为什么？

是为了确保符号位正确扩展，并避免符号位丢失的问题。这是因为带符号整数使用补码表示，右移时要确保保持符号的正确性

23. 浮点数运算

（1）了解浮点数加减法的基本要点：对阶、尾数加减、尾数规格化

（3）四种舍入方式：就近舍入、朝+∞方向舍入、朝-∞方向舍入、朝 0 方向舍入。重点了解就近舍入的规则（舍入为最近可表示的数（包括舍入为偶数））。

（4）浮点数溢出的判定：阶码溢出。

溢出是指结果超出了浮点数表示范围的情况。对于浮点数，溢出通常发生在**阶码溢出**的情况下，意味着计算结果的阶码超过了浮点数表示的最大阶码值。溢出会导致结果变为“**无穷大**”，或者在某些情况下会变为“**非数（NaN）**”，这取决于计算的具体情况

- 阶码的最大值是 255（即 e=255），此时表示的阶码为 E=255-127=128。
- 如果计算后的阶码超过了 128（即阶码值为 129 或更大），则发生阶码溢出，结果变为“+∞”或“-∞”

（5）了解浮点运算中“大数吃小数”现象:浮点数运算不满足加法结合律。

浮点数加法不满足结合律，主要是因为浮点数在计算中涉及到阶码的对齐和尾数的舍入。具体来说：

- 在浮点数加法中，**尾数的精度有限**，因此当数值差异较大时，较小的数的影响会被忽略或丢失，特别是在计算过程中存在舍入误差时。
  浮点数的表示方式（标准 IEEE 754）有限制，在数值较大或较小时，精度会丧失，从而导致不同的加法顺序得到不同的结果。

24. 爱国者导弹定位错误问题：理解造成错误的根本原因

爱国者系统中的**定时器**（时钟）中的浮点数存在了**微小的定时误差**。这个误差在长时间运行时逐渐累积，最终导致了定位和计算误差

25. 如何求一个正数 x 的相反数-x 的补码？

用 x 的补码全部取反，加一

第三章 程序的转换及机器级表示（会看汇编就行）

1. 什么是微程序、微指令、机器指令？机器指令和微程序是什么关系。
2. 描述机器指令的执行过程：取指、译码、执行。
3. 了解 IR、IP 寄存器的作用。
4. 了解机器指令的一般格式：操作码+操作数
5. IA-32 指令系统：操作数类型、几种基本寻址方式、常用指令（汇编指令）（了解）
6. 什么是有效地址、线性地址？
7. 了解 Linux 32 位线性地址空间的划分：用户空间、内核空间

   用户空间的分布：只读代码段、读写数据段、堆、栈、共享库
8. 应用程序的栈区从哪里开始、向什么方向生长？函数的栈帧由哪个寄存器指示栈帧底、又由哪个寄存器指示栈帧顶？
9. 调用者保存寄存器和被调用者保存寄存器各有哪些？为减少被调用函数的准备和结束阶段的开销，应先使用哪些寄存器？
10. 过程调用中栈和栈帧的变化：设 P 为调用过程，Q 为被调用过程，正确描述 P 调用 Q 的过程中 P 和 Q 的栈帧变化（保存调用者保存寄存器、参数准备、CALL 命令的执行、Q 建立自己的栈帧、保存被调用者保存寄存器、Q 开辟临时工作区和对局部变量的操纵、入口参数的获取等，重点掌握 call、ret、push ebp、leave 等指令）。
11. C 语言两种参数传递的方式：传值和传地址，从机器级解释这两种参数传递方式的不同，以及被调用函数在获取参数及获取参数后对值参/变参操纵方式上的不同。
12. C 语言选择结构的机器级表示

    （1）if-else 语句的机器级表示：cmp+jmp

    （2）switch 语句的机器级表示：跳转表。

    （3）如何利用跳转表实现 switch-case 语句的跳转？
13. C 语言循环结构的机器级表示
14. 为什么说递归程序的时、空效率较差？以递归和迭代实现的 int nn_sum(int n)为例说明其机器级的根本原因。
15. 逆向工程：例题和相关习题
16. 数组的分配和访问

（1）数组元素的寻址：基址+比例变址。基址寄存器、变址寄存器存放什么数据？比例因子代表什么？

（2）分配在静态区和栈区的数组分别怎么寻址？

17. 结构体数据的分配和访问

（1）结构体成员如何寻址：基址+偏移

（2）结构体数据作为函数的入口参数，在传值和传地址两种参数传递方式下有什么不同？哪种效率高？

18. 联合体数据的分配和访问

（1）联合体数据有什么特点？各数据成员的首地址都等于什么？

（2）如何对联合体成员寻址？

19. 数据对齐

（1）什么是数据对齐？为什么要数据对齐？

（2）了解交叉编址的基本原理。

（3）简单数据类型的数据对齐策略是什么？

（4）结构体数据的对齐策略是什么？正确计算不同对齐策略下一个结构体数据的字节长度。通过调整数据项的顺序优化结构体数据的存储。

20. 什么是缓冲区溢出？造成缓冲区溢出的根本原因是什么？

是指在程序运行时，向一个缓冲区（如数组或内存块）写入超过其分配空间的数据，导致溢出到相邻的内存区域。在写入栈里面的局部变量，通常是数组的时候，编译器不会对空间的合法性进行检查，读入的数据如果过长，会覆盖掉栈里面原来的合法数据。

21. 什么是缓冲区溢出攻击？简单的缓冲区溢出攻击的基本原理是什么？结合实验 lab3（阶段 1~阶段 4）掌握相关内容。

22. 了解缓冲区溢出攻击的防范措施。

1. **栈随机化（Stack Randomization）**：

- 也称为地址空间布局随机化（ASLR，Address Space Layout Randomization）。

- 每次程序运行时，操作系统随机选择栈的基地址，使得攻击者难以预测栈中特定变量的地址。

- 这种随机化增加了攻击者利用已知攻击代码的难度，因为他们需要在每次攻击时重新计算偏移量。

- 这种方法可以防止某些类型的缓冲区溢出攻击，特别是那些依赖于特定内存布局的攻击。

2. **栈破坏检测（Stack Smashing Protection）**：

- 也称为金丝雀（Canary）技术。
- 在局部变量和返回地址之间插入一个特殊的值（金丝雀）。
- 如果缓冲区溢出发生并覆盖了局部变量，金丝雀值会被改变。
- 程序在函数返回前检查金丝雀值，如果发现被篡改，则触发异常处理，阻止攻击代码的执行。
- 这种方法可以在攻击发生时及早发现并阻止攻击，保护返回地址不被篡改。

3. **非执行内存（Non-executable Memory）**：

- 将栈和堆标记为不可执行，这样即使攻击者成功在这些区域插入了代码，CPU 也不会执行它们。
- 这是一种硬件和操作系统级别的保护措施。
- 当程序尝试执行这些区域的代码时，硬件会触发异常，操作系统会终止该程序。
- 这种方法有效地阻止了攻击者利用缓冲区溢出来执行任意代码，是防止代码注入攻击的有效手段。

第四章 程序的链接（在讲位置的时候一定记得是有每一个其他地方指定的偏移量决定 除了老大哥 elf 头）

1. 了解从源程序到可执行目标程序的转换过程：预处理、编译、汇编、链接等各阶段的输入和输出各是什么？

   预处理输出.i 文件 编译输出.s 汇编文件 汇编输出.o 二进制文件 链接输出可执行文件 器械是 cpp ccl as ld

2. 有哪三类目标文件？

可重定位文件 可执行文件 共享目标文件（可以在两种情况下使用：一是链接器使用其与其他可重定位文件和共享目标文件链接，产生新的目标文件；二是动态链接器将几个这种共享目标文件与可执行文件相结合，作为进程映像的一部分。在 Windows 上称为 DLL 文件，在 Linux 上称为.so 文件。）

3. 什么是可重定位目标文件？产生于哪个阶段？内部编址有什么特点？

可重定位文件是汇编或者编译输出的文件，产生于汇编或者编译阶段。内部编址分了好多的节 首先试试 ELF 头 然后是 text rodata data bss symtab rel.data rel.text debug line strtab 节 多个节分为四个段 分别是代码段 数据段（data and rodata） 符号段 和重定位表（rel.data rel.text） 可重定位文件必须有节头表，包含了每个节的信息，包括大小和名字之类的东西（其他两种可以不要），节头表为节的尾部

4. 什么是可执行目标文件？产生于哪个阶段？内部编址有什么特点？

可执行文件是可以直接执行程序的文件。可执行文件，生成在链接阶段。内部的编址是 •1 绝对地址，包含了每个程序的入口，在没有 rel 段之外有不同的段组成，段是有可重定位文件的节合并组成的，英文名字不变。他们必须有程序头表，在段的开头，指示寄存器如何创建程序的映像（将节的信息映射到唯一的虚拟地址空间里面）

5. 链接器的主要工作是什么？

连接器的功能是符号解析和重定义。将每个模块中引用的符号与某个目标模块中的定义符号建立关联。（也就是说 在每一个目标文件里面的每一个函数都提供了一个入口地址 但是没有内容 在其他模块的地址才有内容 这里就是重定位这个入口地址）

每个定义符号在代码段或数据段中都被分配了存储空间，将引用符号与定义符号建立关联后，就可在重定位时将引用符号的地址重定位为相关联的定义符号的地址

6. ELF 格式的全称是什么？ELF 目标文件格式有两哪种视图？

可执行和可链接格式

- 链接视图：什么是节？可链接目标文件由不同的什么组成？有哪些主要的节？

节是目标文件中的一个逻辑区域，存储特定类型的数据或信息。由节头表和多个节组成。节的类型看上面

- 执行视图：什么是段？可执行目标文件由不同的什么组成？有哪些主要的段？

是程序加载到内存后的一种逻辑区域，用于存储程序运行时所需的代码和数据。段是操作系统加载程序到内存时的组织方式，直接影响程序的执行，由多个节构成，但是是映射到虚拟内存空间的入口。由不同的段和程序头表组成。

7. ELF 头

（1）了解 ELF 头包含的主要内容

（2）对比可重定位目标文件和可执行目标文件的 ELF 头，二者主要有哪些不同？

ELF 头位于文件的开头，是整个文件的核心部分。它描述了文件的组织结构，包括文件类型、机器架构、入口点地址，以及其他信息。所有操作系统在处理 ELF 文件时，都会首先读取 ELF 头以确定如何加载和执行该文件

区别：e-type 一个是 1 一个是 2 入口点地址一个是 0（没有入口点）一个是程序入口（虚拟地址空间）一个又程序头表偏移量一个没有 一个有节头偏移量一个没有 一个有程序头表内容数量一个没有 一个有节头表索引一个没有 一个以段储存一个以节储存

7. 什么是节头表？了解节头表数据结构各数据项的含义。关注.bss 节的特点。

节头表，包含了每个节的信息，包括大小和名字之类的东西（其他两种可以不要），节头表为节的尾部。.bss 是一个逻辑节，只是显示不占内存，只有在程序运行的时候栈内存而且初始化为 0

（也就是说 其实我们的节里面是储存数据内容的 所以节占了内存 当然还有其他信息）

8. 什么是程序头表？了解程序头表数据结构各数据项的含义。为什么有些段的 FileSize 和 MemSize 大小不同？解释其原因（主要针对.bss 节的数据）。

程序头表，在段的开头，指示寄存器如何创建程序的映像（将节的信息映射到唯一的虚拟地址空间里面）.大小，地址，权限，虚拟地址和物理地址等。.bss 等未初始化数据段需要的内存（p_memsz）大于其文件中的表示（p_filesz）。因为他在文件里面没有内存，运行时放在内存里面就有了而且在内存里面还会有更大的内存对齐要求

9. 什么是静态共享库？静态共享库由什么组成？创建共享库文件的命令是什么？C 语言的标准静态共享库叫什么名字？

静态共享库是一种包含可以被多个程序重用的函数集合的二进制文件，主要用于在程序

运行时动态链接,后缀为.a,由多个.o 文件组成。gcc -c myprocl.c myproc2.c
ar rcs mylib.a myprocl.o myproc2.o。叫 libc.a 也就是 stdio.h

10. 链接分哪两步进行？

符号解析和重定位

12. 符号解析

（1）什么是符号解析？符号解析的对象是什么？编译器将符号的相关信息保存在目标文件的哪个结构里？

将每个模块中引用的符号与某个目标模块中的定义符号建立关联。每个定义符号在代码段或数据段中都被分配了存储空间，将引用符号与定义符号建立关联后，就可在重定位时将引用符号的地址重定位为相关联的定义符号的地址。四种符号，全局外部静态标准库，在 symtab 和 dymsym 里面一个是静态的一个是动态的

（2）什么是符号表（.symtab）？了解符号表数据结构各数据项的含义。对用 readelf -s 读出来的某目标文件的符号表，能够正确识别其中各个符号的名称、类型、位置（所在节、偏移）、大小等

是一种数据结构，储存每一个符号的信息比如大小，类型，名称，所在节

（3）什么是 Global symbols、Local symbols、External symbols？对一个具体的程序，能够正确指出其中各符号的类型。链接器主要对哪两类符号进行处理？

第一个问题意思很明确。对外部和全局。

（4）什么是定义符号和引用符号？对一个具体的程序实例，能够正确指出其中各符号是定义符号还是引用符号。

简单

（5）什么是强符号定义和弱符号定义？一般程序中哪些位置出现的符号是强符号定义或弱符号定义？对一个具体的程序实例，能够正确指出其中定义的符号哪些是强定义的、哪些是弱定义的。

强符号是函数名称和已定义的全局，弱符号是没有定义的全局

（6）链接器对单一定义的符号的解析规则是什么？

在单个文件里面说话。引用的符号必须有定义。重复定义的符号会报错，局部变量不是符号，优先解析强符号，外部符号不处理未定义会报错。Static 不是强符号，他只对当前文件负责，但是是一个符号

（7）链接器对多重定义的符号的解析规则是什么？对一个具体的程序实例，能够正

确写出符号解析后符号定义和符号引用之间的关联关系，及因此而造成程序中的变量之间关联性改变，对出现的显式或隐式错误能够指出产生问题的原因。

三点。不允许出现相同的强符号，允许出现相同的弱符号而且选一个，弱符号和强符号同名选强符号，而且注意是定义的时候的要求，一个声明一个定义完全可以

（9）能够概述链接器符号解析的全过程

- E、D、U 三个集合中分别存放什么对象？

E 里面是可重定位文件，D 里面是已经定义的符号，U 是还没有找到定义的符号

- 对一个具体的程序实例，概述其符号解析的过程，重点描述 E、D、U 三个集合的变化

对一个程序的编译命令来说，先是一个可重定位文件进来，丢进 E，修改 D 和 U，注意一个文件可重定义的话也肯有符号解析导致的 D 和 U 的改变，如果进来一个归档文件尝试用 U 里面的符号和她进行解析，成功了就修改 U 和 D。直到两者不在变化。此时，E 里面的文件被简单丢弃，继续处理

- 对静态共享库的操作：其模块被使用或不被使用是怎么判别的？使用到的模块怎么处理？没有被使用到的模块又怎么处理？

遍历目标文件和静态库的符号表，逐一解析未定义符号。 如果某个目标文件引用了库中某个模块的符号，并且该符号在之前的解析中未定义，链接器将标记该模块为“被使用”。如果模块中定义的符号没有被任何地方引用，则该模块被认为“不被使用”。（这里的定义是有无其他文件重定义了 意思和 c 语言一样）。被使用就提取出来加载到模块，未被使用狙跳过丢弃

（10）为什么符号解析成功与否与命令行中文件的顺序有关？举例说明。

13. 重定位

（1）重定位过程：1）合并相同的节、2）对定义符号进行重定位、3）对引用符号进行重定位，各做什么具体工作？

第一个是把什么 data，rodata，和 text 合并。重定位是把一些函数的入口和变量的内容定位到定义他们的地方。解析符号，地址结算，修改重定位表里面的地址，一般会抛弃。

（2）链接器怎么知道目标文件中有哪些位置需要重定位？编译器将重定位信息记录在目标文件的什么结构中？

通过重定位表。放在 rel.data rel.text rel.rodata

（3）什么是重定位信息表？了解重定位信息表数据结构各项的含义。.rel.data 节和.rel.text 节各保存哪种数据的重定位信息。

是目标文件中的一个数据结构，用于记录需要修改的地址信息。保存要重定位的 data 和 text 内容（外部函数和全局变量的引用）

（4）两种重定位类型：R_386_PC32 和 R_386_32

- R_386_PC32 重定位：
  - 相对寻址方式下，有效地址如何计算？
  - 什么是重定位的初始值？
  - R_386_PC32 方式下，如何计算重定位值？
- R_386_32 重定位：R_386_32 方式下，有效地址等于什么？

（4）对于一个具体的程序实例，能够根据每处的重定位信息计算重定位（地址）值

首先用 text 的地址用偏移量 offest 计算重定义位置，然后用符号位置加上固定位置减去重定义位置（这个固定位置其实是因为 eip 指向下一条指令不是当前的入口的差值）

绝对引用只需要用目标地址加上固定地址就可以了，把一个多字节地址直接加载在入口的 call 处

14. 了解可执行文件执行时的加载过程。

通过调用 execve 系统调用函数来调用加载器

加载器（loader）根据可执行文件的程序（段）头表中的信息，**将可执行文件的代码和数据从磁盘“拷贝”到存储器中（实际上不会真正拷贝，仅建立一种映像，这涉及到许多复杂的过程和一些重要概念，将在后续课上学习）**

加载后，将 PC（EIP）设定指向 Entry point (即符号_start 处)，最终执行 main 函数，以启动程序执行。

作业（新版书习题）：

所有布置过的作业 + 适当扩展（如奇数题、偶数题）

实验：

lab1、lab2 前四阶段、lab3 前四阶段

第四章 程序的执行

1. CPU 执行指令的过程是怎么样的？简述

CPU 执行一条指令的过程可以分为以下几个步骤：

①取指令。从 pc（eip）里面取出相应的指令到指令寄存器 ir 里面

②指令译码。对 ir 里面的指令进行译码，翻译出控制信号

③源操作数地址计算并取操作数。根据寻址方式确定源操作数地址计算方式，若是存储器数据，则需要一次或多次访存；若是寄存器数据，则直接从寄存器取数后转到下一步进行数据操作

④执行数据操作

⑤目的操作数地址计算并存结果。

⑥下条指令地址计算并将其送 PC。

2. 什么是流水线的运行方式？简述

一条指令的执行过程可被分成若干个阶段。每个阶段都在相应的功能部件中完成。如果将各阶段看成相应的流水段，则指令的执行过程就构成了一条指令流水线。进入流水线的指令流，由于后一条指令的第 i 步与前一条指令的第 i+1 步同时进行，从而使一串指令总的完成时间大为缩短。

3. 系统吞吐率的计算公式是什么

1/（时间周期*10^(-12)）

4.为什么流水线模式需要再每一个操作后加一个寄存器储存数据，为什么

每一个操作的时间周期要设置成一样的值？

因为要在不同操作阶段隔离数据和传递数据。因为要让多个数据在同一时间进行程序的不同阶段并且顺利传递数据。

4. 什么样的指令适合用流水线？

指令长度尽量一致，有利于简化取指令和指令译码操作

指令格式尽量规整，尽量保证源寄存器的位置相同。

采用 load/store 型指令风格。指令集中只有 load 指令和 store 指令能访问存储器，其他指令一律不能访问。可以保证除 load 和 store 指令外的其他指令在执行阶段都不访问存储器，有利于减少操作步骤，以规整流水线。

数据和指令在存储器中要“对齐”存放。有利于减少访存次数，使所需数据在一个流水段内就能从存储器中得到。

5. CICS 指令系统有什么特点？

指令系统复杂，指令长而且格式多，寻址方式多，指令传唱度可变，地址可变，时间周期长，不同指令之间时间周期差异大，难以显式优化，而且会输出显式的指令码

6. RISC 的指令有什么特点？

①指令数目少

②指令格式规整

③采用 load/store 型指令设计风格

④采用流水线方式执行指令

⑤采用大量通用寄存器

⑥采用硬连线路控制器

⑦实现细节对机器级程序可见

但是编译工作量大

第五章 层次存储器的结构

1. 存储器有哪些分类？

寄存器 静态 RAM RAM Flash 存储器 硬盘 光盘

2. 按存取方式分类有哪些分类？

随机存储器 顺序存储器 直接存储器 相联存储器（记忆：按照难不难找来的）

随机存储器是地址寻址，时间级别是常数级别 叫做 RAM

顺序存储器按顺序存放盒取出，如磁带 SAM

直接存储器首先通过地址定位到入口处，再用顺序查找如磁盘存储器 DAM

相联存储器 CAM,必须知道内容的特征进行查找，如快存

3. 按信息的能否读写分类？

读写寄存器（RAM 芯片）和只读寄存器（ROM 芯片）

4. 按断电后的能否恢复进行分类？举例？

非易失性寄存器（ROM，磁盘，光盘），易失性寄存器（RAM，cache）

5. 按功能分类？

高速寄存器：由静态 RAM 组成，在主存和寄存器之间

主存：程序运行的时候指令直接面向的寄存器，CPU 执行程序的时候必须把储存地址变成主存地址

辅助寄存器：磁盘存储器，和主存交换数据还才才能被 cpu 访问

海量后备储存器

最后两个叫外存

6. 存储器的层次结构是怎么样的？简述 cpu 如何从内存中提取指令

Cpu 到寄存器到 cache 到主存到硬盘到光盘。速度越快，则容量越小，越靠近 CPU

CPU 可以直接访问内部存储器，而外部存储器的信息则要先取到主存，然后才能被 CPU 访问

CPU 执行指令时，需要的操作数大部分都来自寄存器

当需要从存储器中存取数据时，先访问 cache,如果不在 cache 中，则访问主存，如果不在主存中，则访问硬盘

此时，操作数从硬盘中读出送到主存，然后从主存送到 cache。

数据使用时一般只在相邻两层之间复制传送，而且总是从慢速存储器复制到快速存储器。

传送的单位是一个定长块，并在相邻两层间建立块之间的映射关系

7. 什么是时间局部性和空间局部性？出现局部性的原因是什么？为了利用局部性，我们如何控制数据的传递加速效率？

时间局部性指的是某一个单元在被访问后很短时间内再次被访问，空间复杂性指的是某一个单元被访问之后，它邻近的单元很快被访问。原因是数据在主存中是顺序储存的，他们的地址是连续的，指令也通常是顺序储存的，常常被重复访问。我们一般会把访问单元和它最近的一个单元一起作为一个块提取出来，作为一个块一起放到 cache 里面

8. 什么是 cache？他的存在如何提高 cpu 的运算速度？

cache 是一种小容量快速存储器，由多块 SRAM 组成，它的运算速度几乎和 cpu 是一个数量级的。在主存和 cpu 之间设置 cache，把主存里面频繁访问

的数据放在 cache 里面，这样的话 cpu 没有必要每次都像主存提取数据，能直接快速地从 cache 里面提取数据

9. cache 的工作原理是怎么样的？

为了便于 cache 和主存之间交换数据，我们按照块的模式在 cache 和主存空间都划分为内存单元一样的块进行数据的传递，这样的块叫做主存块。Cpu 执行指令的时候，它先在 cache 里面查看有没有需要的指令或者数据，如果有直接提取进行读写，如果没有，在主存中寻找，找到后以主存块的模式传递到 cache 里面，再进行读写

10. cache 的访问时间是如何计算的？

在访问过程中，cpu 首先要判断数据或者指令在不在 cache 里面，如果在，那么时间就是访问数据的时间 t1，假设命中率是 p，那么不命中率就是 1-p，那么此时 cache 要从主存里面寻找数据并且写会 cache，假设这部分时间是 t2，再传回 cpu，时间为 t1，那么时间开销就是 t1+t2. 总的计算公式是 p*t1+（1-p）*（t1+t2）=t1+（1-p）*t2. 由于局部性，命中率几乎是 1，时间平均值几乎是 t1

11. 我们程序员应该如何编写程序以让程序的效率达到最大？

首先，循环要用内循环（行优先），如果可以，展开循环。循环过大，可以每次运行四个运行单元（利用流水线）

第六章 异常和中断

1. 什么是中断？

中断是一种使 CPU 终止正在进行的程序，转而调到处理特殊事件的操作，处理结束后，又跳到正常的断点初继续向下执行指令

2. 中断的过程是怎么样的？简述

中断请求，中断响应，中断处理，中断返回

3. 中断过程有哪些部分？

中断源，中断请求，中断响应，中断系统

4. 异常和中断的相同点和不同点？

异常是 cpu 的错误处理，中断是外部事件的处理

5. 异常的分类？

故障：CPU 某些错误发生的中断

陷阱：程序中主动用 INT 指令产生的中断

终止：执行指令发生严重错误，导致程序无法进行

6. 内中断和异常的关系？内中断的处理特点？

异常包括内中断和外中断。内中断转指的是由程序内部产生的异常，外中断是硬件导致的异常。内中断的错误类型在代码中有所体现，而且不受 if 中断指示符的影响。在我们的学习中，异常是指 cpu 内部的中断

9. 程序是如何发现和处理异常的？

在程序执行过程中，cpu 发现了一个某种预定义条件，要终止该程序而产生了一个错误中断信号，进而调用异常处理程序对异常进行处理

10. 常见的故障类型有？

除于一个 0，缺页，或者数组越界，常常在指令进行前产生或者指令执行的时候产生。

11 陷阱是什么？

在执行引起异常的指令之后，把异常情况通知给系统。

对于软中断之类的陷阱异常，实际上就是产生异常信号指令之下的一条语

句的地址。

所有软中断，就是在程序中写了中断指令，执行该语句就会去调用中断处理程序，中断处理完后又继续运行下面的程序。

软中断调用与调用一般的子程序非常类似。借助于中断处理这一模式，可以调用操作系统提供的很多服务程序。

另外一种常见的陷阱异常是单步异常，用于防止一步步跟踪程序。

11. 终止有什么特点？

中止是在系统出现严重问题时通知系统的一种异常；

引起中止的指令是无法确定的；

产生中止时，正执行的程序不能被恢复执行。系统接收中止信号后，处理程序要重新建立各种系统表格，并可能重新启动操作系统；

中止的例子包括硬件故障和系统表中出现非法值或不一致的值；

12.中断的类型有什么？

Cpu 检测的内中断，如除法除以 0

程序检测的内中断，如程序的软中断,单步中断

外设导致的可屏蔽中断(INTR)，如 I/O 设备的中断

外设导致的不可屏蔽的中断（NMI），比如硬件故障，掉电，存储器错误，总线奇偶校验错误等

12. 外中断的特点？

由 8259a 提供，或者自制电路提供。受到中断位置的标志位 IF 影响（IF 决定可屏蔽中断是否执行），可屏蔽中断一次可以处理多个中断

13 优先级如何？

Cpu 内中断>不可屏蔽中断>可屏蔽中断>程序内中断（单步中断）

14 什么是中断向量表？

中断向量表就是各种中断的处理程序的入口地址表中断类型有 0XFFH 种，每一个占四个字节，比如 INT 4AH 就是中断地址为 4*4AH。实模式下大小为 1kb，起始位置是主存物理地址 0，包括 ip 和 cs 的首地址

15. 中断类型号是如何获取的？

除法错，单步中断，不可屏蔽和断电中断，溢出中断由 cpu 提供

软中断 INT n，从程序运行里面读出类型号（中断指令的下一条的地址

外中断，标准外设 8259a 提供，非标准的从自制电路通过接口提供给总线

16. 如何把中断指令传入中断向量？

注意，中断向量是主存的 1k 绝对地址：

例如：Mov $p1,0x45*4

Mov %cs,0x45*4+2

第一个是 p1 函数的地址就是 eip，第二个是 cs

17 如何取出中断向量里面的中断指令？

Mov 0x45*4，%ax

Mov 0x45*4+2，%dx

17. 保护模式下的中断向量表是这样的？

保护模式下，它叫做中断描述符表（IDT），按照统一的描述符内容定义表项。抛弃了 eip 和 cs，我们加入了 eip 入口地址和类型和权限，8 个字节，2kb。他有一个寄存器 IDTR，表面他的 pa（物理地址）.他的具体内容是高 IDT 错误类型 16 位主存偏移值，段选择符 16 位，段偏移量 16 位，一个门属性（权限），一个对齐没用的字节

18 软中断指令有哪些，具体什么操作？

INT n (sp)是栈指针

(FLAGS)->(SP) 把 0 同时送到 IF,TF（flags 是 if，tf，保护现场）

（CS）->(SP) （4*n+2）->cs

（ip）->(sp) (4*n)->ip

其实和程序跳转一模一样

IRET

Sp->ip

Sp->cs

Sp->FLAGS

第七章 I/O 处理

1. I/O 子系统的输出是如何输出的？

程序利用高级语言的 I/O 标准库函数实现外设 I/O，通过陷阱指令转化为系统内核执行

2. I/O 子系统的分类有什么？

I/O 子系统包括 I/O 软件和 I/O 硬件，I/O 软件有用户空间 I/O 软件和内核空间 I/O 软件，I/O 硬件由内核空间 I/O 软件控制

用户空间的 I/O 软件包括用户程序中的 I/O 请求，运行时的系统，内核空间的 I/O 软件包括与设备无关的 I/O 软件，设备驱动服务程序（设备商自己提供的功能程序），中断服务程序

3. 设备驱动和中断处理的关系？

外设种类众多，中断处理为所有设备设置了抽象部分，由操作系统实现，和基于设备不同的具体部分，由设备驱动实现。抽象的系统功能调用，会将请求传递到这些具体的中断程序上来

4. I/O 设备有什么分类，什么是 I/O 总线？

由机械部分和电子部分组成，机械部分是 I/O 设备本身，电子部分则称为设备控制器和 I/O 适配器。I/O 总线用来在系统中链接多个设备控制器和总线桥接器

5. 什么是 I/O 端口？

设备控制器中有许多数据寄存器，控制寄存器和状态寄存器，他们用于存放外设与主机交换的数据，状态和控制信息，他们一起叫 I/O 端口，其中数据寄存器叫数据端口，状态和控制寄存器叫状态/控制端口

9. 为什么要 I/O 端口寻址？又哪几种寻址方式？有多少个寻址空间？

因为要对 I/O 端口和外设进行快速寻址，I/O 端口寻址包括独立编址和统一编址两种，前者对每一个 I/O 端口单独编号。IA-32 允许有 64k 个 8 位端口和 32k 个 16 位端口

10. 什么是 I/O 指令？

在底层 I/O 软件中，可以通过将控制命令传到控制寄存器控制外设工作，可以读取状态寄存器获取外设工作状态，可以直接访问数据寄存器进行数据的的输入和输出，这些操作的指令就叫 I/O 指令

11. 有哪些 I/O 指令？

（1）输入指令 IN 放入 al 和 ax（16 位和 32 位）

形式：IN $PORT,%AL IN ($PORT+1,$PORT),%AX IN %DX,%AL IN %DX,%AX

（al 和 ax 确定读哪个端口的内容）

（2）输出指令 OUT(寄存器输出到外设寄存器)

OUT %AX,$PORT(两个八位字节，还送到了 PORT+1)

端口地址超过 255，放在 dx 里面？

INS,OUTS,处理字符串的，其他和上面一样

12. 无条件传送的特点有哪些？

不查询外设状态直接用 IN 和 OUT 传送指令，必须保证外设和 CPU 之间在传送数据的时候有相同的速度（避免数据丢失，缓冲区溢出和促进数据流水线高速传递）

13. 有条件传送的特点？

查询外设状态，外设准备好再传送数据，适合慢速外设和 CPU 之间

14. 什么是中断传送方式？有什么特点？

CPU 启动 I/O 外设后，不再等待外设启动完成，而是转而执行其他程序，当外设准备好了之后，向 CPU 发出中断请求，CPU 接受请求后停止当前程序，转而用很短的时间处理外设指令后，回到程序继续运行

特点：提高了 CPU 的效率，可以处理突发事件，提高了计算机运行的灵活性

15.什么是直接储存器（DMA）传送方式？

适用于高速外存和 CPU 的数据传递。在端口上用通道技术实现，在外设准备好数据传送和 CPU 准备好数据接受后，外设向 CPU 发出一个 DMA 传送信号，CPU 让出总线控制权给外设，让外设可以在短时间内和主存成批传递数据

往年题：

1、 对 C 程序进行编译生成 Release 版程序时，编译器会做一些优化工作。请举出 5 个优化场景，简要说明做了什么优化，以及为什么能加快执行速度。（10 分）

使用寄存器作为循环计数器；

使用寄存器作为数组下标，进行变址寻址；

使用寄存器传递参数；

使用带比例因子的变址寻址操作数，获取 EA，代替算术运算；

使用宽字节类型的寄存器，进行数组元素的操作，减少循环次数；

使用顺序访问数组元素的方式，代替循环访问数组元素；

(1) 指出该段程序执行效率不高的原因 （2 分）。

使用局部变量作为循环计数器；使用局部变量作为数组下标，进行变址寻址；

在条件表达式中，对同一个变量 buf[i]，进行了两次赋值

1、什么是中断和异常？两者有何差别？什么是中断描述符表？中断和异常的响应过程是什么？（10 分）

中断和异常：一个进程在执行过程中，正常的逻辑控制流被特殊的事件所打断，CPU 转到处理这些事件的内核程序去执行，从而引发一个异常控制流。

中断一般是指 CPU 之外的事件发生，如按键盘，鼠标等，是与当前正在执行的指令无关的异步事件。异常是 CPU 正在执行的指令引发的事件，如除 0、访问数据的地址超出程序空间范围等等，是与正在执行的指令相关的同步事件。异常分为故障、陷阱和终止。

中断描述表（中断矢量表）：中断服务程序和异常处理程序的入口地址构成的一个表，每个表项占 8 个字节，存放入口偏移地址及相应代码段的描述信息；这些表项是按照中断和异常的编号（即中断和异常的类型号）顺序排列的。

中断和异常的响应过程：CPU 的控制逻辑确定检测到的中断和异常类型号，从中断描述表取出对应的表项，计算相应的处理程序的入口地址，保存当前的 CS、EIP、EFLAGS 等信息，然后转到处理程序的入口地址对应的位置去执行。在中断处理程序中有 IRET 指令，执行该指令从堆栈中恢复 CS、EIP、EFLAGS 等，从而回到被中断的位置继续执行。对于三类异常：故障、陷阱和终止的处理策略有所差异。故障表示能够修复，引起故障的指令会被再次执行；陷阱则是执行引发陷阱的下一条指令；而终止就要结束程序的运行

2、 设一个函数中有语句 int temp=global; 其中 global 是一个初值为 35 的 int 类型全局变量。编译器对 global 的定义(int global=35;)和 temp=global 编译时，分别会在可重定位目标文件中的哪些节生成哪些信息？（10 分）

对于 int global=35; 在 数据节(.data) 节存放 35（0x00000023）； 在 字符串节(.strtab), 存放字符串 “global”; 在符号表节（.symtab），存放有关符号 global 的信息，包括定义该符号的节号、在相应节（.data）节的地址、数据长度（4 个字节）、属性信息等。

对于 int temp=global; 在代码节(.text)中， 生成相应的语句，其中 global 的地址（偏移量）用占位符 0x00000000。在代码节的重定位节(.rel.text)，要记录 代码节的相应位置（global 占位符的起始地址）要被什么符号（符号表的哪一项）、用什么定位方式（地址相对程序计算器 PC 的 32 位偏移）所代替，还包括计算地址时的附加项等。

2、 举例说明编写 C 程序或者编译器优化时利用相应原则进行优化的做法（包括优化前的方法，优化后的方法）。（10 分，每小题 2 分）

① 提高 CPU 中 cache 的命中率

任务：二维数组求累加和

优化前；二重循环，按列序优先访问

优化后：二重循环，按行序优先

② 提高 CPU 中指令流水线的利用率

任务：一维数组求累加和。

优化前：使用循环，逐个元素相加

优化后：循环展开，消除循环，变成一系列加法语句

③ 使用 CPU 中处理速度更快的指令

任务：表达式计算，将一个数 *9

优化前：用乘法运算，乘 9 运算

优化后：左移 3 个二进制位运算（相当于乘 8），再加原来的操作数

④ 使用 CPU 中单指令多数据流指令或串操作指令

任务：两个数组相加，得到一个新数组

优化前：用循环的方法，逐个对应元素相加

优化后：成组运算，一次多对数据同时运算，减少了循环次数

另一个例子：将一个数组中所有元素置 0。优化前，用循环的方法，逐个元素置 0。优化后，

用 memset 函数实现，该函数的实现封装了串操作指令。

⑤ 使用多核 CPU 中多线程处理能力

任务：两个数组相加，得到一个新数组

优化前：逐个对应元素相加

优化后：创建两个或者多个线程，每个线程完成数组中部分元素（一些行）的求和，即按行将数据分成几块，每个线程完成一个块的相加操作。

3、 在一个程序的运行过程中（如正在执行一个二维数组求累加和），用户按了键盘上的某个键，计算机系统会做出哪些响应（即一系列的处理过程）？中断分哪几类？请举例说明各类中断在何种情况下产生。（10 分）

实模式下，键盘产生一个**中断请求**。

系统中断逻辑电路，将中断请求转为**中断号**，发送给 CPU。

CPU 响应中断请求。首先保存正在执行程序的当前 eip 和其他寄存器的内容；然后根据中断号**查询中断向量表**，获取该中断相应的中断处理子程序的入口地址；然后中断当前程序，**跳转到该中断处理子程序中**，处理键盘输入；执行完中断处理子程序后，取回保存的 eip 和其他寄存器，**返回到原程序**中继续执行。

中断类型包括：

不可屏蔽外部中断：例如电源掉电。

可屏蔽外部中断：例如外部设备产生的中断请求。

CPU 检测的内部中断：例如除法出错。

程序检测的内部中断：例如程序中的软中断调用。

4、 可重定位目标文件中有哪些节？各节中主要有什么信息？什么是符号解析？链接的过程是什么？（10 分）

可重定位目标文件中有：

.text 节：编译汇编后的代码

.data 节：已初始化的全局变量、静态局部变量

.rodata 节：只读数据

.bss 节：未初始化的全局变量、静态局部变量；初始化为 0 的全局变量、静态局部变量。

此外还有文件头、节头表。

符号解析，是将每个模块中引用的符号，与某个目标模块中的定义符号建立关联。

链接的过程包括符号解析和重定位两方面。符号解析如上所述。重定位则是分别合并代码和数据，并根据代码和数据在虚拟地址空间中的位置，确定每个符号的最终存储地址，然后根据符号的确切地址来修改符号的引用处的地址。

错题：

右移和除法不是永真的·，比如-1，但是左移是永真的

32 位平台 long 是 4 个字节

```
int isPositive (int x)
{
    return !(!(x^0)|((x>>31)&0x01))
}
```

参数变量入栈是从右向左入栈右边的在大地址，左边的在小地址

注意看 obj 反汇编代码里面断点的位置 breakpoint in getbuf 不是 getbuf 首地址停下来 是前面还有一个地址

Rw 是可读写段 filesize 是文件大小 mensize 是内存大小 后者大于前者 原因是.bss 在文件里面没有空间，在内存中可以映射出一段空间

重定位里面代替重定位符号的是 R_386_PC32 这种东西

重定位前的值就是 call 后面的字节内容

注意函数首地址对齐的导致的最终首地址

就计算重定位地址的时候相对于首函数地址的偏移值就是这个什么 R_386_PC32 字符串出现的位置的偏移量

考虑强弱符号！！！