个人感悟：将整个计算机知识体系捋清，成为计算机发展史，在发展史的主干上上下求索。不要追求简单，多写点，多记点，后面到简的时候自然简。

第一课

直播：大黄蜂--学生端

录播：鹏程万里加密软件。上完课的第二天上传阿里云，下载之后，根据老师给定的授权码绑定机器使用。

道场：github。

课堂纪律：

- 不要人身攻击、黄色、政治、课堂无关的。

如何学习

- 修仙：仙气、基于仙气，修炼自己的成长路线
- 游戏：新手村、基于新手装备+教程、刷副本、升级、转职。

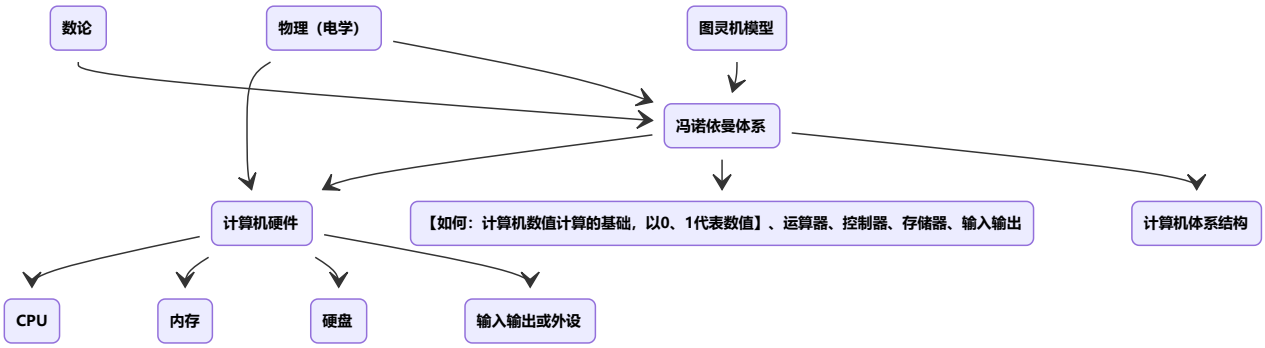
抓住重点：推理。

先定方向，再看书，坚持。

计算机的种子：

- 需求，希望有一个输入到机器，得到我们想要的结果。输入 --> 输出。
- 图灵：得到计算机的简易版（接口）。冯诺依曼将图灵的简易版实现，得到冯诺依曼体系。如：【伏羲开天，周文王】

计算机一切由 0 1组成，万事万物由阴阳组成。



注意老师的学习流程：

1. 图灵机，冯诺依曼对图灵机进行实现，得出冯诺依曼体系。其中体系中最主要的点是，以0、1代表数值。
2. 那如何用来表示0和1呢？物理（电学）、数论。

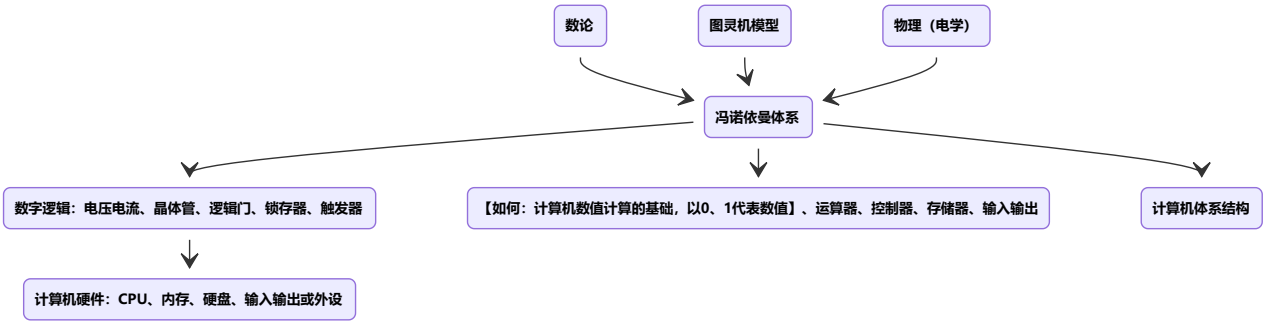
> 为什么不用*10*进制，用*10*进制所需状态要*10*个标识。通过香浓公式计算得到，计算机表现较优的状态是*2*、*3*。最重采用的是*01*两种状态表示。这与伏羲的阴阳类似，阴阳即万物。

3. 01用电压表示，解决了信息的表示之后。那如何处理信息呢？就是信息的运算，则需要运算器。有了运算器之后，那如何调节运算的顺序？以及运算的行为呢？从而要控制器。
4. 能运算了，且能控制运算行为了，那对于运算的结果需要保存，从而需要存储器。
5. 存储了之后，你要给人看，需要渲染吧？从而有了显示器，即输入输出。
6. 继而：物理（电学）＋冯诺依曼体系＋数论 ＝ 计算机硬件。
7. 计算机体系结构：更小的单元、更多的逻辑、更高的性能、更低的消耗。

易经原名变经，改名为了易经。

1. — 阳
2. -- 阴

计算机体系结构书本，辅助画图。



计算机的种子：图灵机，类似 易经 的 1。

图灵机由一生二，二声四，四生八，八生万物。

学习路径：根学习法。根是不变，根据根衍生出来的新东西，都可以根据根来推理学习。当我们学了一个叶，如Java，我们可以根据Java接入的底层操作逻辑，来推理Go之类的学习。

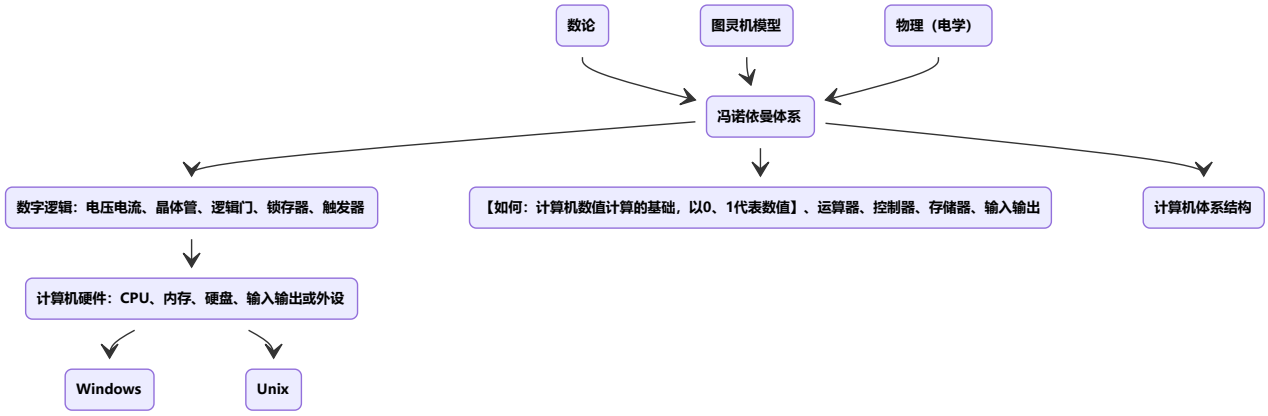
第二课

图灵机是假设，并没有落地，冯诺依曼体系落地实现。冯诺体系若干部件由数字逻辑组成，构成了计算机硬件：CPU、内存、外设、硬盘。

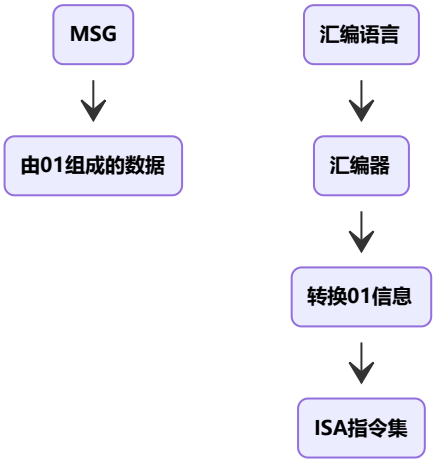
伏羲八卦，万物生。

为什么e接近3：$2 < e < 3$，八卦为什么不为3，西方三只脚符号。易经是3，易经讲究变，偶数对称，不具备变化，而基数具有变化性，从而取为3，通过3两两组合，乾卦、坤卦等八卦，得到先天八卦，八卦相互演变得到64卦象。

一划开天，得出阴阳。



汇编语言与机器语言的发家史



> 汇编语言由编译期实现*01*转换，*C*面向汇编语言编程即可。

因为CPU是控制器，所以需要控制计算机，就需要控制CPU。如何控制？CPU厂商必须给出一堆二进制01的组合，来告诉我，如何取操作CPU。这套组合称为：指令集架构（ISA）。

由于PC端用得较多的为：Intel，从而准备了Intel的开发手册给到各位，后面也是基于Intel的开发手册来学习。

此节课我们从上一节推出的 计算机硬件组成 和 计算体系结构，我们可以很简单的猜想到研究此学科是 《计算机组成原理》，然后阅读：前言。

计算机组成原理前言

> *21*世纪是科学和技术奇迹频出的时代。计算机已经做到了人们期望它做到的一切——甚至更多。生物工程解开了细胞的秘密，使科学家能够合成*10*年前无法想象的新药。纳米技术让人们有机会窥探微观世界，将计算机革命与原子工程结合在一起创造出的纳米机器人，也许有一天能够植入人体，修复人体内部的创伤。普适计算带来了手机、*MP3*播放器和数码相机，使人们彼此之间能够通过*Internet*保持联系。计算机是几乎所有现代技术的核心。本书将阐述计算机是如何工作的。
> 从*20*世纪*50*年代起大学就开始教授这门被称为计算的学科了。一开始，大型机主导了计算，这个学科包括对计算机本身、控制计算机的操作系统、语言和它们的编译器、数据库以及商业计算等的研究。此后，计算的发展呈指数增长，到现在已包含多个不同的领域，任何一所大学都不可能完全覆盖这些领域。人们不得不将注意力集中在计算的基本要素上。这一学科的核心在于机器本身：计算机。当然，作为一个理论概念，计算可以脱离计算机而独立存在。实际上，在*20*世纪三四十年代计算机革命开始之前，人们已经进行了相当多的关于计算机的科学理论基础的研究工作。然而，计算在过去***40***年里的发展方式与微处理器的崛起紧密联系在一起。如果人们无法拥有价格非常便宜的计算机，*Internet*也无法按照它已有的轨迹取得成功。
> 由于计算机本身对计算的发展及其发展方向产生了巨大影响，在计算的课程体系中包含一门有关计算机如何工作的课程是非常合理的。大学里计算机科学或计算机工程方向的培养方案中都会有这样一门课程。实际上，专业和课程的认证机构都将计算机体系结构作为一项核心要求。比如，计算机体系结构就是*IEEE* 计算机协会和*ACM*联合发布的计算学科课程体系的中心内容。
> 介绍计算机具体体现与实现的课程有各种各样的名字。有人将它们叫作硬件课，有人管它们叫作计算机体系结构，还有人把它们叫作计算机组成（以及它们之间的各种组合）。本书用计算机体系结构表示这门研究计算机设计方法和运行方式的课程。当然，我会解释为什么这门课程有那么多不同的名字，并会指出可以用不同的方式来看待计算机。
> 与计算机科学的所有领域一样，计算机体系结构也随着指令集设计、指令级并行（*ILP*）、*Cache*缓存技术、总线系统、猜测执行、多核计算等技术的发展而飞速进步。本书将讨论所有这些话题。
> 计算机体系结构是计算机科学的基石。例如，计算机性能在今天的重要性超过了以往任何时候，为了做出最佳选择，即便是那些购买个人电脑的用户也必须了解计算机系统的结构。
> 尽管绝大多数学生永远不会设计一台新的计算机，但今天的学生却需要比他们的前辈更全面地了解计算机。虽然学生们不必是合格的汇编语言程序员，但他们一定要理解总线、接口、*Cache*和指令系统是如何决定计算机系统的性能的。

通过前言，可以很顺利的推理出，计组 与 嵌入式 以及 计算机体系结构有很大的联系，于是我们可以找到对应的数目，进行目录的比较，更甚至是内容的比较，发现其实大部分都是一样的（耦合又分离，你中有我，我中有你），只不过在中间加入了部分章节用于讲述特定的内容，而这些相同的内容呢，正好在 **CSAPP** 书中有。所以 先看了 **CSAPP** 很多东西你都能知道一些，可以用于构建轮廓，由此往外进行分支，做到枝繁叶茂。

讲完了硬件，那么怎么来操作硬件呢？

1. 怎么写指令去操作呢
    a. 因为CPU是控制器，所以需要使用控制计算机就是通过 01 组成的指令去控制CPU，而这些指令则是由硬件厂商规定。
    b. 01代码是机器码，但是如果我们这样去操作就会导致编写程序可读性很差，维护性也很差，编写也面临很大困难。 所以就有了 汇编语言 与 汇编器
2. 写了存在哪儿？
    a. 存在硬盘上，做持久化存储
    b. 但是CPU的处理速度 大幅度的超过了 磁盘 的IO速度，如果都从里面读写，就会导致CPU大部分时间处于等待中
    c. 于是我们就添加缓存 ----> 首先是内存， 但是内存还是很慢呀 ----> CPU 中嵌入 缓存
    d. 为何不直接使用大块的缓存？ ---- 成本问题
3. 讲到磁盘和内存以及缓存，就有 RAM 和 ROM
4. a. RAM：random access memory 随机存储内存，可读可写，掉电易失
    b. ROM：read-only memory 只读内存，持久存储
    c. 可以干嘛呢？？？ ---> 想一下，我们程序中 代码文件是不可写的，只读的，而数据则是需要 存取的，所以硬件条件较为苛刻的情况下，如微小型的设备，我们就可以使用 ROM 存储代码，只用少量的RAM 存储运行时需要更新更改的 数据即可。--》在这里就可推出嵌入式。

第三节

汇编语言 --> 汇编器 --> 转换为 --> 01机器码 --> CPU读取【硬盘--> 内存-->缓存-->CPU】

对内存中的+1操作：

1. 将数据加载到寄存器；
2. 对寄存器中的数据加1；
3. 将寄存器中的数据写回内存。

CISC：因一条指令包含多条微指令，从而对于汇编和机器而言，其压力相应较小，但增加了CPU压力。

RISC：精简指令集。一条指令一个微操作。其压力在汇编，汇编需要将代码翻译成非常多的指令操作，CPU并没有那么多压力。

为什么先有CISC，后有RISC。指令正交，即理论来说每条CISC指令都是相互独立的，但是因为CISC里边包含的多条微操作有些表示的含义是重复的，表明微指令冗余了。

同时有一个比较大的问题是，当CISC指令集发生变化的时候，汇编器都需要进行升级，且CPU也需要发生变化，劳动力比较大。

RISC阻屏较RISC的高，其时延相对较低。

英特尔的CPU架构：组合了RISC和SISC。如MOP是CISC，uOP是RISC。

汇编语言：

1. 操作CPU，去控制其他硬件完成操作。
2. 具象化：操作CPU的控制单元，控制存储单元，运算单元去对来自缓存的数据进行CRUD操作。

汇编，可以看作是面向指令集编程，指令集操作是用于CPU硬件，所以可以看作是汇编是面向CPU编程的，即CPU面向机器/硬件编程。

为何汇编 会 扯到 计算机组成 ？ ？ ？ ？ 搜索汇编相关书籍的时候，出来了一本《计算机组成及汇编原理.pdf》书本。

> 计组与汇编为什么会有联系呢？因为在知识书上，汇编面向指令集，指令集面向硬件，则汇编是与组成有关系的。
>
> 或者说，在知识数上，汇编是计算机的硬件的上层，而计组是讲述计算机如何运作的，所以两者是有联系的。

汇编语言是面向指令集的，而指令集又是计算机硬件厂商提供的，计算机的硬件是 计算机组成和计算机体系结构 共同研究的内容，所以 汇编 书籍中也存在一些章节是讲计算机组成、计算机体系结构的！

在了解汇编作用后，学习汇编内容：《计算机组成及汇编语言原理**.pdf**》、汇编语言**--**基于**X86**处理器。

- 在计组和计算机体系上面介绍了JVM。
- JVM是基于RISC的指令集开发的，JVM也可以看做是汇编语言，JVM有汇编指令。

看**408**的书籍

道法自然：遵循事物发展的规律，上下求索，即道法自然，天人合一。

Intel和ARM的内核看哪个的？Intel指令是RISC，且其指令更接近人能理解的指令。

总结

课上了三节，我觉得已经到了阶段性里程。我觉得需要总结一下，但是在总结的时候，如何能够体现老师的学习思路，上下求索、推理，的方式进行讲解。

> 还是没有抓住脉络。

首先是，在计算机开篇初期，人类的需求：希望输入一段内容，机器能够输出所想要的。该需求由图灵提出了一个猜想，其供后继者加以实现，实现团队中的主角是冯诺依曼。那对于冯诺依曼体系需要哪些东西呢？

1. 机器能够识别的信息。那用几态来表示？二进制？十进制？哪个更好？香浓理论。01
2. 机器能够识别之后，你希望计算机对这信息进行怎样的处理？对计算机进行加？减？还是其他处理状态？控制器。
3. 那能控制信息进行处理了，那对信息的表达处理，得要一个元件来实现？那就是计算器。
4. 那对于计算的结果，需要保存吧？那就是存储器。
5. 单单保存了，我都没有看到效果，并没有任何作用。那就需要输入输出，即外设。

对于其所需要的东西，可以由什么来得到呢？物理（电学）、数论（离散）。这就得到了一堆的硬件：控制器、计算器、存储器、外设。对于这一堆硬件，也有生态，如电压电流、晶体管、逻辑门等组成这些硬件。这个生态也有专门学科去研究壮大，计算机组成原理。

当然，对于冯诺依曼体系，也有人去进行研究设计，追求更小单元、更高性能、更高存储等目标，这个就是计算机体系结构。

在这些基础上，通过查看书本找到计算机组成原理和计算机体系结构的共性（通过书本前言和目录总结），同时对于应用，有了嵌入式开发。

从而有了三朵花：计算机组成原理、计算机体系结构、嵌入式开发。这三朵花是基础，他们都有共性，即硬件。

> 对于学习这三朵花，以*csapp*为主，然后看其他书本的特性东西为辅助。

有了01表示信息之后，我们需要对现实世界语言和机器能够识别的01做映射，即编码。对于这编码的规则是由大佬定制的，名为ISA，其由两个指令集：CISC和RISC。

- 刚开始出现的是CISC，复杂指令集，即一条CISC指令可以被CPU解释为多条微操作指令。因指令正交的原因，导致多条CISC指令可能包含相同的一条微操作指令，从而存在冗余。同时，CISC指令对于CPU开发者而言，是一个非常不友好的，他的一条指令新增，对于CPU开发者就需要增加新的劳动力。

- 因CISC的弊病，出现了RISC，精简指令集，即一条RISC指令就表示一条微操作指令。这样对于CPU开发者而言，当只有RISC指令标准有新增的时候，才需要改动，但变动范围就比较小，工作量降低。

有了ISA指令集后，虽然可以让机器处理了，但是让人直接编写01组成的指令，是非常没有人性的，从而有了对ISA指令进行封装的需求，即面向ISA开发的汇编语言，当然有了汇编语言，就需要汇编器，将汇编指令翻译为机器能识别的01指令。

有了汇编之后，基本上，面向机器开发的工作就固定了。

后面就出现了一堆面向人类编程的语言，即对汇编语言进行封装，汇编语言写了汇编器，c语言面向汇编开发产，经由汇编器编码为汇编。

此时三朵花就变为了四朵花：计算机组成原理、计算机体系结构、嵌入式开发、汇编语言，这四朵花是基石，不变的四点。计算机后续生万物，都从他们开枝散叶出去了。

> 对于汇编语言的学习：《计算机组成及汇编语言原理.*pdf*》、汇编语言--基于*X86*处理器

第四节

如何设计一门语言：面向汇编设计语言。

汇编面向CPU编程。

ISA指令集组成：key-value。

1. 指令组成：操作码 + 操作数。
    a. 你这条指令是用来干什么的？你要完成什么工作？
    b. 以及你对哪个数字进行操作。

看汇编的定义：《汇编语言 基于x86处理器.pdf》。

内存最小寻址单元，最小为多少？1byte，8bit。为什么？看书本：计算机体系结构与组成原理。

1. 不能过小，不然频繁存取，如最小寻址为1bit，那读取完一个数据就需要读非常多。
2. 不能过大，因为会有多余的bit，如16byte，过大了。

面向汇编设计语言：

1. 查看内核的汇编代码，了解内核的类型系统，屏蔽对内存单元大小的访问。如定义int、double、float、long、double、boolean，用于表示访问/操作多大的内存单元。
2. 抽象对数据的操作：加减乘除操作，屏蔽mov sub add等指令。
3. 添加语言自身的特性。如：`int a = 1;。

C语言：最纯粹的汇编抽象，自身并没有扩展语言自己的特殊属性。从而他们说C语言最简单，最纯粹。

> *C*++语言：在*C*语言的基础上，对*C*语言添加了自身语言特性，如面向对象。

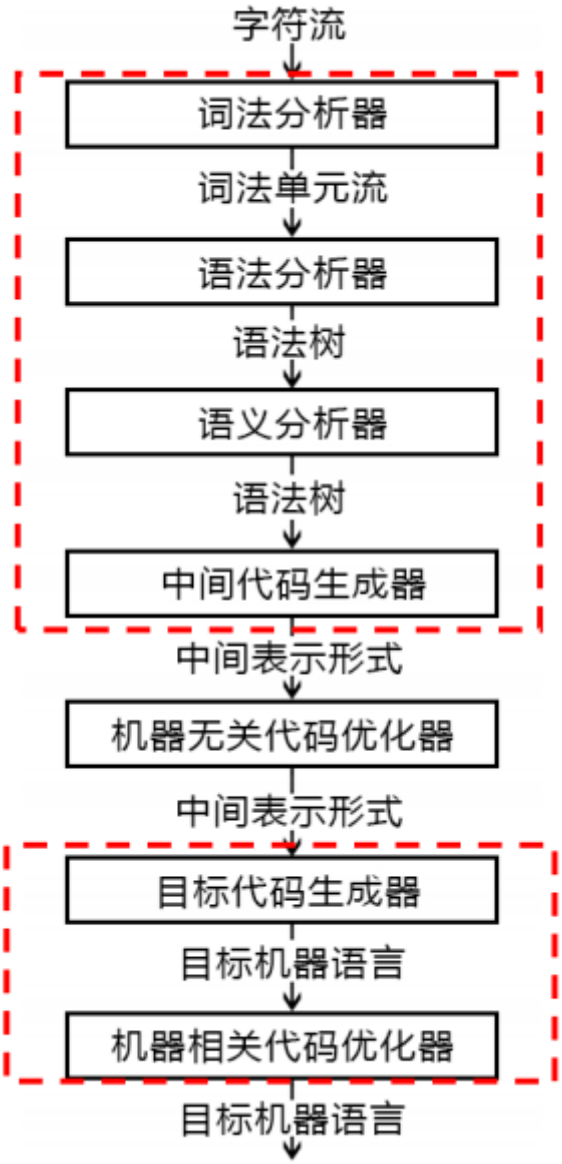
而C语言到汇编语言，需要一个编译器，该编译器是怎样做的？汇编器到CPU，又是怎样完成的？这部分的研究就是由编译原理完成的。即编译原理研究的是将一个语言转换为另一个语言。

- PPT来源于哈尔滨工业大学的 编译原理。JIT：将编译器与汇编器打包了。

翻译与编译的区别：Java编译成字节码，将字节码再翻译成机器码（JVM中使用宏，通过汇编器翻译成机器码）。

编译器：通过编译器，可将源程序编译成汇编语言，或直接编译为机器码。

第五节



对于一台机器而言，下边的目标机器语言是不变的，也就是 后端部分是不会改变的，如果面对很多的上层语言来说，是不是都需要编写整个的编译流程呢？

其实是不必的，只要让上层的高级语言都产生同样形式的 中间产物，后端逻辑就可以复用，只需要根据自身语言的定义和特性 来将语言转换为中间产物即可。

编译原理：讲述数据结构了来源与出现。

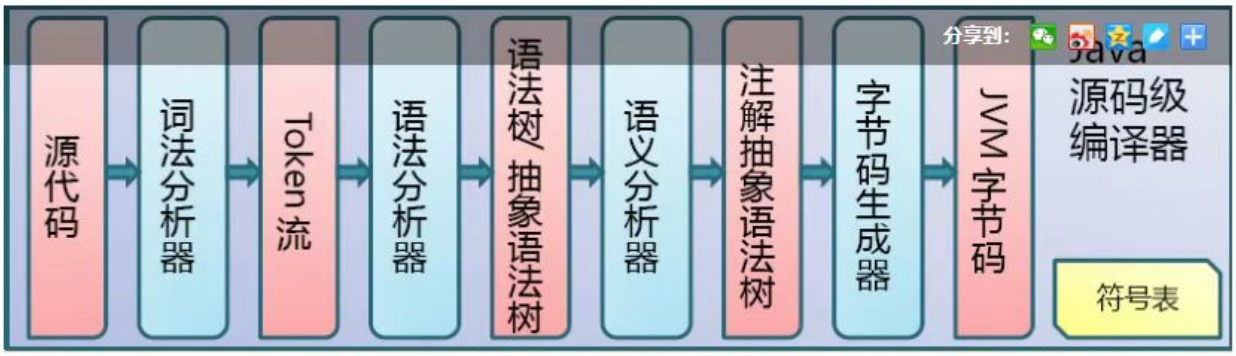
数据结构：定义了存储数据的形式；算法定义了对数据的操作。这两个一般是在一起出现的。

> 书本：数据结构C语言 清华大学

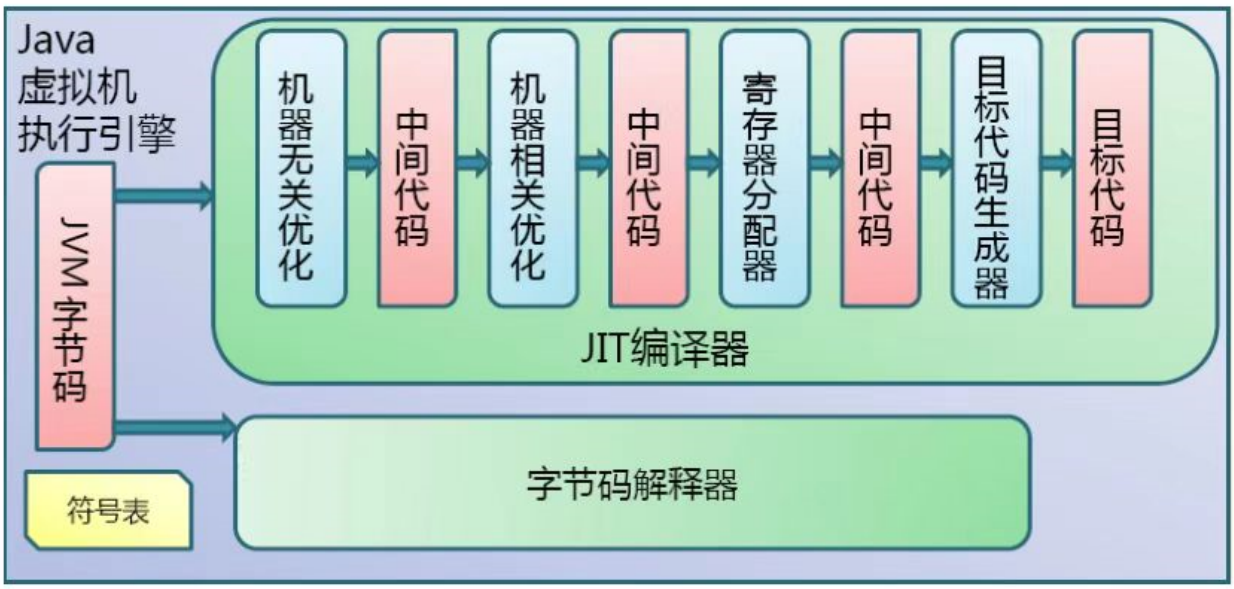
正是因为语言提供的特性，影响着语言最终走向到哪一步。

Java 代码编译和执行的整个过程

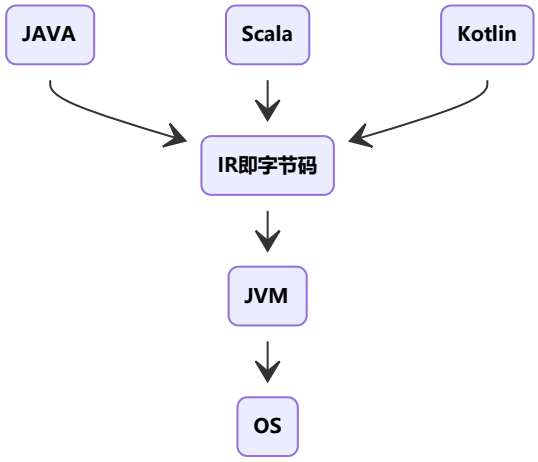
Java 代码编译是由 Java 源码编译器来完成，流程图如下所示：



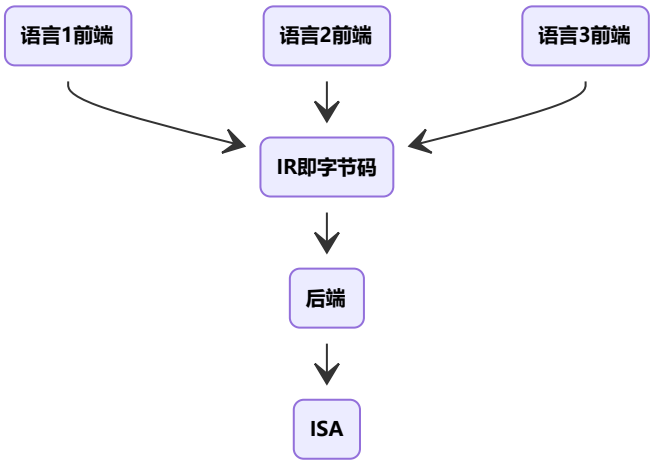
Java 字节码的执行是由 JVM 执行引擎来完成，流程图如下所示：



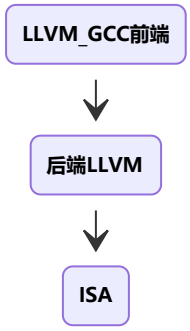
第六节



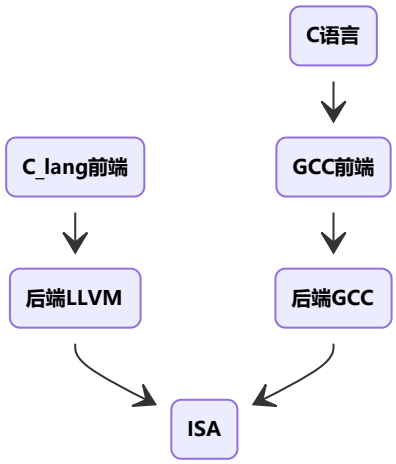
IR与后端相关，GCC有自己的IR，LLVM有自己的IR，从而GCC与LLVM是竞争关系的。



Object-C Swift



LLVM_GCC是前期妥协用的工具，苹果后续对LLVM_GCC前端进行改造，得到了C_lang前端，解析情况如下：



> 学习思路：新的东西出来，必须满足：已存在的东西的所有功能，且需要增加自己的新的特性，这样才具备竞争力。所以研究新的东西时，可以基于现有的东西的了解 程度 + 新特性 来进行学习和选型。

GCC全称GNCC，C语言的建立取决于GCC前端。实际上一切语言的特性均面向 编译器 来设计和编写的。

GCC的缺点，没有模块化，没有解耦合。

而LLVM为了面向各语言进行编译，实现模块化，热插拔。一切皆模块化则可进行高度可定制，开源后，所有语言均可按照模块化机制接入。

ebpf的原理：

由C/S架构引入了VM，而ebpf是在OS上的，所以OS借鉴C/S的VM，也可在OS内核中嵌入了VM，实现对应用程序的IR处理，但因为linux个人的原因，想在OS中添加VM非常困难，所以利用原有的VM，即BPF【用于过滤应用程序的数据】，进行了功能扩展，得到了ebpf。
对于LLVM和BPF的理解，我们可以通过书本来佐证《LLVM编译器实战.pdf》，通过前言来佐证。

总结：

因为LLVM都是模块化开发，所有东西都是可热插拔。

LLVM IR链接器：不同的文件通过LLVM编译都需要链接，所以需要IR链接器。

第七节

名词推理：

碎片化的知识，了解树干，知 道（一切皆推理）。

控制求知欲，跟着老师先把主干建立起来。不用急着打点。

编译原理：

1. 研究语言从语言到词法的过程，称为词法分析，得到词和词性；
2. 分析词和词性，得到句子结构【语法树】，这个过程称为，语法分析；
3. 对 语法树 进行 语义分析，得到IR（每个人都有自己的理解，即IR各异）；
4. 对IR进行 翻译/优化，得到目标语言。

名词解释：

编译器、汇编器、解释器、**JIT**、**GCC**、**LLVM**、**BPF**、**GraalVM**、**C1/C2**、**javac**、链接器

编译器：

第八节

回顾：汇编解释器、c++解释器，JIT（编译为目标平台的机器码）、编译器。

JIT：即时编译器，将源代码直接执行为目标代码，并缓存起来，后续使用的时候直接拿来使用。从而JIT需要消耗更多的时间，以及空间。

解释器：逐行解释为目标代码执行，不缓存。从而对于解释器而言，消耗较少的时间和空间。

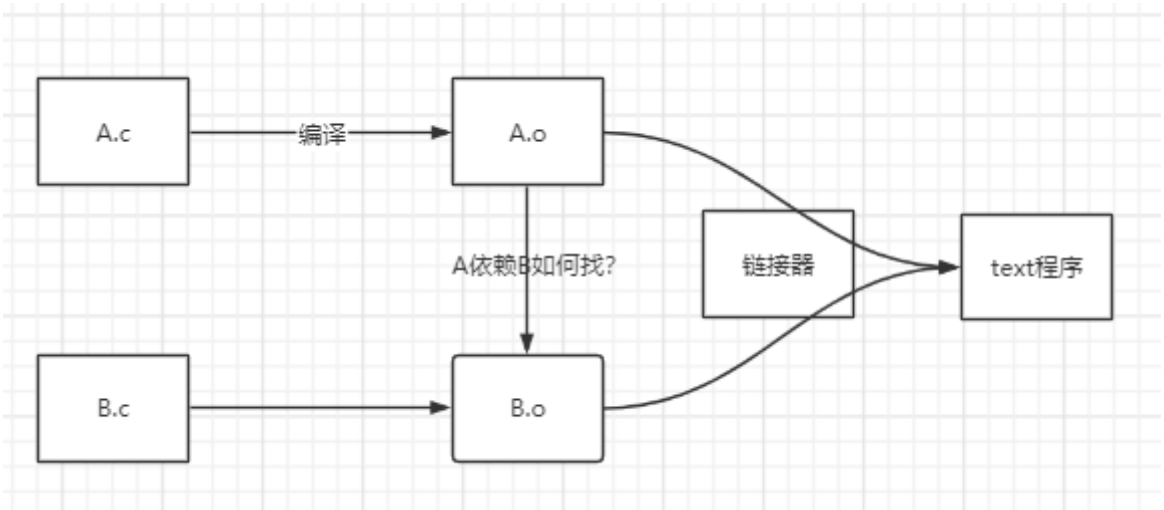
> 那什么时候使用*JIT*，什么时候使用解释器进行跑代码呢？这个就需要你代码进行分类了，分为多样性高的代码和多样性低的代码。
>
> 对于多样性高的代码，丰富度高，差异性大，此时适合使用解释器。因为如果你采用*JIT*，则每段字节流都是不同的，执行*A*段需要开辟空间，需要消耗更多时间，当执行完*A*段后，需将代码占用空间销毁；再执行*B*段代码，重复*A*段的执行过程，此时采用*JIT*的性价比就低，适合采用解释器来执行，逐条解释快【因为代码的多样性高，所以重复劳作的情况不存在】，且不占用较多空间，性价比更高。
>
> 而若是代码丰富度低，被一直重复执行，此时就适合*JIT*了。*JIT*一次编译，多次使用。不然使用解释器【多次解释，多次使用】，则不断进行重复劳作，就耗时更多，效率更低，响应就慢，此时采用*JIT*的性价比更高。

剩余名词讲解：GCC、LLVM、BPF、GraalVM、C1/C2、javac、链接器。

C2：面向服务器，C2比C1效率更快。

Javac：java的前端，将java源程序经javac输出为AST，以及将AST转为中间字节码的表示。

链接器：起到代码缝合作用。



云原生：程序员写的代码能够一站式的管理。云原生使得程序员仅考虑CRUD。。。

libjvm.so：是linux的java虚拟机的完整实现。

第九节

LLVM和GraalVM

LLVM：出现的目的是统一编译器；

GraalVM：统一虚拟机平台/解释器，或者是统一VM。如ruby、python、rust语言都需要解释器，但他们自身的解释器比不过HotSpot，从而有了GraalVM的出现动机。

HotSpot出了模板解释器，还有JIT，JIT有三个，C1、C2和Graal Compiler三个。其他解释器都没有HotSpot厉害，从而接入进来。

这节课：语言的本质是什么？对下一级的语言进行高度抽象。

Java是对JVM的抽象；JVM是对真实机器处理指令的抽象，实现跨平台；C语言对汇编语言的抽象，汇编是对机器码的抽象。

若要讨论哪个语言好？需要加上场景，加上上下文。

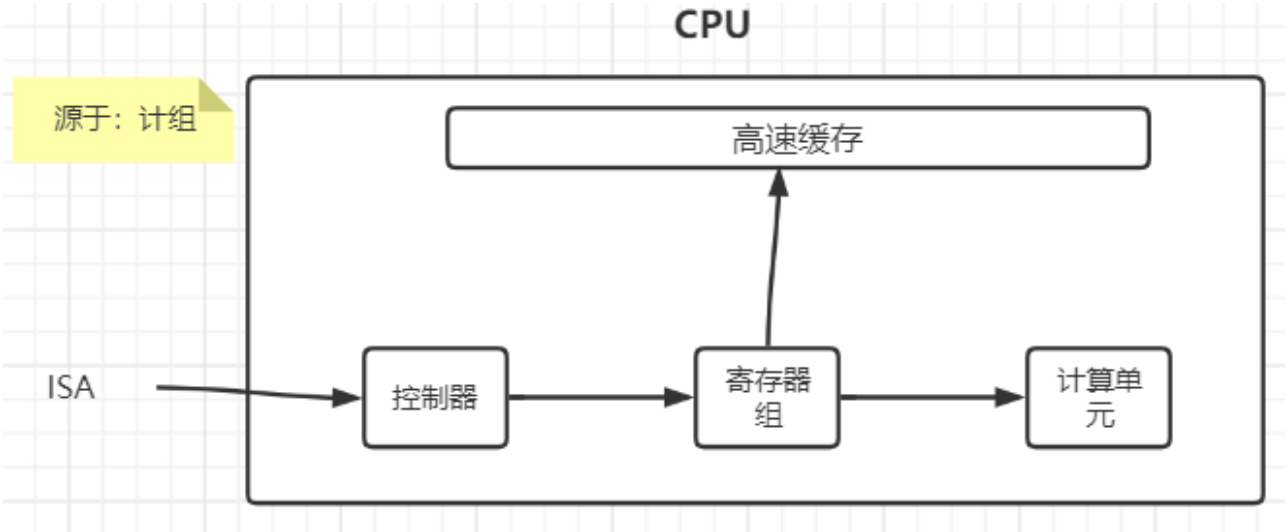
根据实际应用场景，选择合适的语言：选择合适的语言，就是选择语言的语法，因为语言的底层都是一样的。主要是看语言是否使用你要处理的事情的场景，语言的生态又怎样？开发人员的质量是否有要求？

开发者的第一门语言是C语言，因为它是最纯粹的汇编抽象，没有其他的什么语法层面的优势（如：切片、守卫语法，lambda、函数式编程、面向对象。。。）

C语言学不好的原因是：不懂底层，如计组、汇编、编译原理、计体。

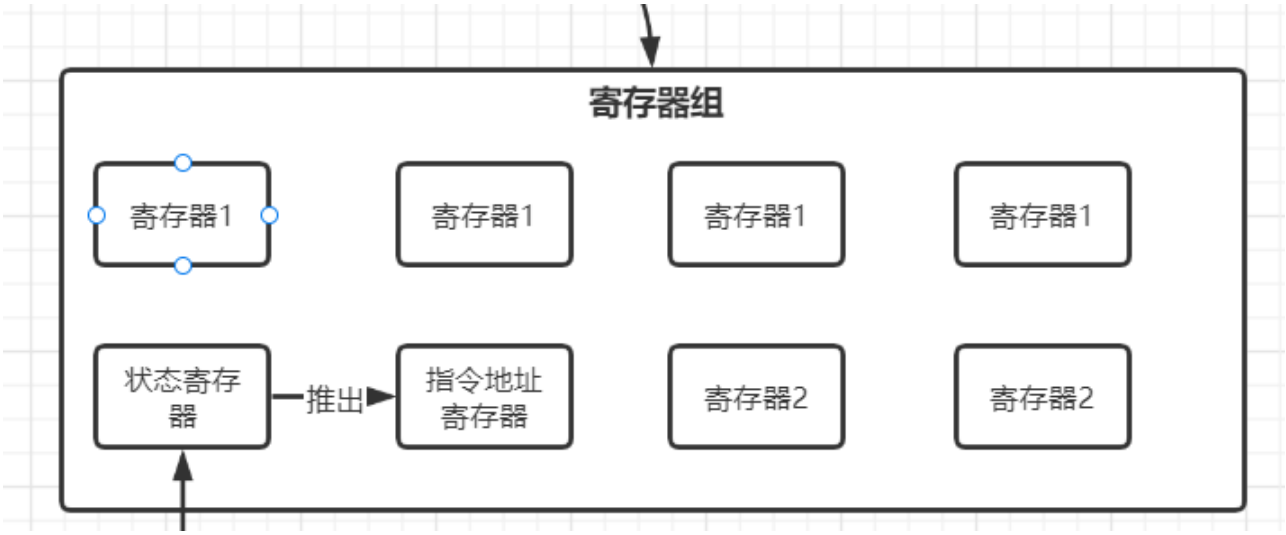
ISA指令集定义了一组规则，这组规则去控制 控制器，将寄存器的数据放入计算单元进行CRUD；其中寄存器的数据从高速缓存中来，计算单元计算的结果也存放到寄存器，由寄存器再写回高速缓存或内存。汇编是控制寄存器组进行运算，写回内存。因为汇编是在内存层面的。

ISA指令集控制控制器，控制器操作寄存器组完成CRUD并写回；汇编器控制寄存器组进行CRUD，并写回内存。



那如何理解汇编是操作寄存器组的呢？

首先寄存器组就是由一组寄存器组成。



那既然是一组寄存器，不同的寄存器完成什么样的功能呢？这个就需要根据图灵机的状态机来决定了。比如对于程序的指令状态，如if...else...，这个是需要跳转的，则需要跳转指令，从而需要一个状态寄存器完成跳转功能，那对于汇编指令而言，为了表示跳转，则需要相应的汇编助记符来完成对状态寄存器的控制。

一般我们的汇编指令是存储在内存中的，那寄存器想要拿到指令，则需要一个指令位置的计数器来记录后续要回到哪条指令，即程序计数器，而对于当前正在执行的指令的内存地址信息，也需要一个寄存器进行存放的，即指令地址寄存器。

> 程序计数器（*PC*，*Program counter*），用于存放指令的地址。为了保证程序*(*在操作系统中理解为进程*)*能够连续地执行下去，*CPU*必须具有某些手段来确定下一条指令的地址。当执行一条指令时，首先需要根据*PC*中存放的指令地址，将指令由内存取到指令寄存器中，此过程称，为*“*取指令*”*。与此同时，*PC*中的地址或自动加*1*或由转移指针给出下一条指令的地址。此后经过分析指令，执行指令。完成第一条指令的执行，而后根据*PC*取出第二条指令的地址，如此循环，执行每一条指令。

> 指令寄存器（*IR*，*Instruction Register*），用来保存当前正在执行的一条指令。是临时放置从内存里面取得的程序指令的寄存器，用于存放当前从主存储器读出的正在执行的一条指令。当执行一条指令时，先把它从内存取到数据寄存器（*DR*，*Data Register*）中，然后再传送至*IR*。指令划分为操作码和地址码字段，由二进制数字组成。为了执行任何给定的指令，必须对操作码进行测试，以便识别所要求的操作。指令译码器就是做这项工作的。指令寄存器中操作码字段的输出就是指令译码器的输入。操作码一经译码后，即可向操作控制器发出具体操作的特定信号。

同时，为了内存维护方便，以及提高内存碎片使用率，一般存储在内存中的指令（如汇编指令）是分段存储的。这样对于汇编指令而言，就涉及到两种指令跳转情况：

1. 内存段内的指令跳转。这个通过状态寄存器实现跳转。
2. 内存段间的指令跳转。这个就需要借助call指令实现，以及当调用完成后，需要返回原来指令的下一条指令，则需要return，简称ret指令。

除此之外，内存中的数据，有些是共享的，有些又是私有的，对于共享的，可能存活时间比较就，对于私有的，可能就一次性就是用就没了。那如何表示共享内存的指令流信息和私有的指令流信息。此时很自然而然的扯到了数据结构。

1. 对于共享内存，因存活时间比较久，无需一次性消耗，则可以使用堆结构存储，无需经常修改。即有了堆内存。
2. 而对于私有内存，因其指令流存活时间比较短，且因为call的调用可能会涉及到不断嵌入的原因，此时栈结构就非常合适，从而就有了栈内存。

第十节

状态寄存器和跳转指令连接在一起，为什么需要呢？图灵机的状态机的存在，相应的也需要跳转。

对于指令段，也称称为代码段。

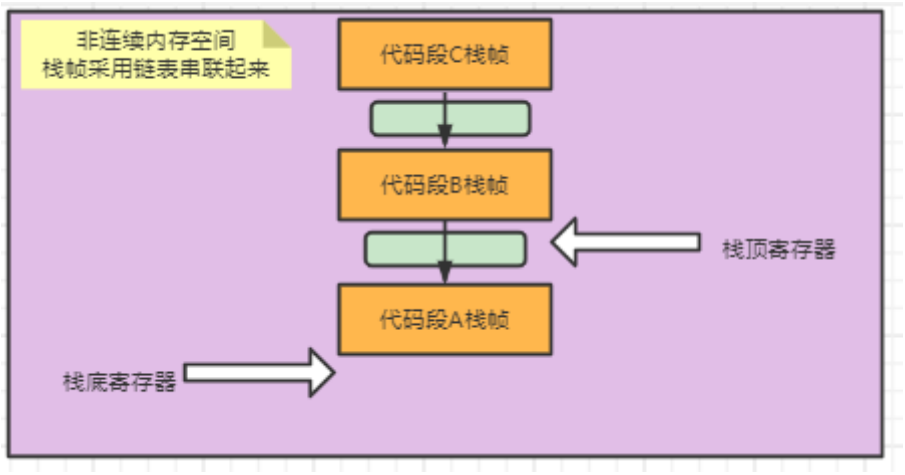
对于堆内存和栈内存，可以挂在到C语言上？？这样就引入了C语言？

应该要进入C语言了。

第十一节

回顾昨天学习内容，将内容知识体系串起来。

由于我们的指令放在内存中执行，需要开辟一个指令流片段的私有数据，采用数据结构中的栈来表示要存储的数据，从而需要一个栈顶和栈底的寄存器来指示栈顶和栈底位置。有了栈之后，对于每个指令流存到栈中的数据用栈帧来表示。那如何从一个指令流跳转到下一个指令流呢？那如何开辟下一个指令流的空间呢？移动栈顶寄存器，将栈顶寄存器的指向地址改变，即开辟一个新的栈帧来存储新的指令流，并移动栈底和栈顶寄存器的地址指向到新栈帧的底部，并留有一个数据空间用来留存上一个栈帧的栈底位置，用于栈底寄存器回退，栈顶寄存器则在栈帧抛出时逐步往下指向即可。对于非连续空间的栈，其栈帧是用链表串联起来的。



如何开辟空间：将栈顶寄存器往高地址移动（链表情况可也可是低地址），即完成栈帧空间的开辟。

将二进制用16进制表示后，这样所使用地址表示数量就少了很多，由32位变为可由8位表示即可。

操作系统是一段特殊区域，将高地址的一片区域专门用来跑操作系统的程序。

栈内存：指令流的私有数据

堆内存：指令流之间共享的数据。

数据内存：保存写在程序中，定义在汇编中的写死的数据

该小结主要讲了汇编代码的实现流程。

主要看书本：《汇编语言：X86处理器》，看本书的目的是看汇编相关的，根据自己的需求看书，直接看3.1。看汇编是因为C语言需要非常严格的汇编基础。根据汇编推断C语言就非常容易了。

3.1小结就是基本上是带术相关的，但也很重要，不然推**C**的时候就很懵了。

MASM：微软汇编

```
main PROC
    mov eax,5
    add eax,6

    INVOKE ExitProcess,0
main ENDP
```

1. `main PROC`和`main ENDP`之间是一段汇编代码，其中这两段之间是指令流。
2. mov：移动赋值；
3. add：加；

Intel开发手册有四卷，那如何看呢？首先看第一章概要，让大家有个概览。

1. 第一卷：发展史、其他章节一点点点
2. 第二卷：所有指令集的说明
3. 第三卷：如何面向CPU编程
4. 第四卷：讲述与指定平台强挂钩的寄存器

其中**2**、**4**用来作为查询手册，**1**、**3**用于详细观看。

先看卷1 chapter2和chapter3的 3.4小结。

学习：论语--学而篇。

第二期第一节

开源想法：在Redis中写个Mysql插件，由Redis写回mysql，且redis和mysql在同一台机器中，这样就没有网络问题了。

虽然redis同步写mysql比单写redis慢，但强一致性对于读还是有好处的。

参考

- 深入浅出让你理解什么是LLVM：https://www.jianshu.com/p/1367dad95445
- 各位 如果想学习 研究 汇编这一门语言，可以学学王爽的汇编语言 ，国内汇编书籍写的最好的，没有之一
- 大家共享笔记：https://docs.qq.com/doc/DSEJyTld2UmVFTHpn