1. 【汇编语言】小甲鱼零基础汇编真正全集1-17章
2. <https://www.bilibili.com/video/av18899713?p=5>
3. Ep 1: 基础知识01; 机器语言; 汇编语言的产生; 汇编指令; 寄存器; 反编译; (2019-7-18)
4. Ep 2: 基础知识02; 汇编指令; 伪指令; 其他符号; 存储器; 指令和数据; 存储单元; CPU对存储器的读写; 地址, 命令, 数据; 总线; 地址总线, 数据总线, 控制总线; (2019-7-18)
5. Ep 3: 基础知识03; 指令和数据; 8086CPU; 地址总线; 64位CPU, OS, software; 1Byte=8bit; 数据总线; 控制总线; (2019-7-18)
6. Ep 4: 基础知识04; 小结; 每一种CPU都有自己的汇编指令集; 指令和数据没有任何区别, 都是二进制信息; (2019-7-18)
7. Ep 5: 内存地址空间; 主板; 接口卡; RAM and ROM; 内存空间地址分配; 将各类存储器看成一个逻辑存储器; 必须要从CPU角度考虑问题; 主存储器地址空间, 显存地址空间, 各类ROM地址空间; 统一的逻辑存储器; (2019-7-19)
8. Ep 6: 寄存器(CPU工作原理); CPU概述; 内部总线; 外部总线; 寄存器概述; 所有寄存器都是16位的; 通用寄存器ABCD; AX; 16位数据在寄存器中的存放情况; 为了兼容性, AX可以分为AH和AL; (2019-7-19)
9. Ep 7: 寄存器(CPU工作原理); 字在寄存器中的存储; 关于数制的讨论; 几条汇编指令, 不区分大小写; 检测点2.1; (2019-7-19)
10. Ep 8: 寄存器(CPU工作原理); 物理地址, 唯一的地址; 16位结构的CPU; 8086CPU给出物理地址的方法, 用两个16位地址合成20位地址; 地址加法器工作原理, 物理地址=段地址\*16+偏移地址; 段地址\*16就是数据左移4位; 第一个比喻; 第二个比喻; (2019-7-19)
11. Ep 9: 寄存器(CPU工作原理); 段的概念; 内存并没有分段, 段的划分来自于CPU; 段的概念; 内存地址单元小结; 同一个物理地址可以有不同的段地址和偏移地址, 条条大路通罗马; 检测点2.2; (2019-7-19)
12. Ep 10: 寄存器(CPU工作原理); 段寄存器; 段寄存器提供段地址; CDSE; CS, 代码寄存器; IP, 指令指针寄存器; CS和IP最关键; 修改CS, IP指令; mov, 传送指令; 转移指令JMP; JMP 2AE3:3; 代码段; CPU只认CSIP指向的为指令; 检测点2.3; (2019-7-20)
13. Ep 11: 寄存器(CPU工作原理); 实验一; (2019-7-20)
14. Ep 12: 寄存器(CPU工作原理); (2019-7-21)
15. Ep 13: 寄存器(内存访问); 内存中字的存储; DS和[address]; mov可以将数据直接送入寄存器, 也可以将一个寄存器中的内容送入另一个寄存器, 还可以将一个内容单元中的内容送入一个寄存器; 8086CPU不支持将数据直接送入段寄存器; 字的传送; (2019-7-21)
16. Ep 14: 寄存器(内存访问); 字的传送; sub指令和add指令相反; (2019-7-21)
17. Ep 15: 寄存器(内存访问); mov, add, sub; 验证debug; 数据段; 问题3.5分析; (2019-7-22)
18. Ep 16: 寄存器(内存访问); 栈; FILO; 入栈; 出栈; CPU提供的栈机制; PUSH; POP; 栈顶; PUSH指令的执行过程; (2019-7-22)
19. Ep 17: 寄存器(内存访问); POP指令的执行过程; 栈顶超界的问题; 栈顶超界是危险的; PUSH和POP指令的格式; 栈与内存; (2019-7-22)
20. Ep 18: 寄存器(内存访问); PUSH和POP指令; 问题3.7; 问题3.8; 问题3.10; 在SS和SP中存放栈顶的段地址和偏移地址; (2019-7-22)
21. Ep 19, 寄存器(内存访问); 栈段; 问题3.11; 问题3.12; 可以将一段内存定义为段, 比如代码段, 数据段, 栈段; 数据段, 段地址放在DS中; 代码段, 段地址放在CS中; 栈段, 段地址放在SS中; CPU将内存中的某段内容当作代码, 是因为CSIP指向了那里; 将某段内容当作栈, 是因为SSIP指向了那里; 关键在于CPU中寄存器的设置, 即CSIPSSSPDS的指向; 检测点3.2; (2019-7-23)
22. Ep 20, 第一个程序01; 引言; 一个源程序从写出到执行的过程; 编写汇编源程序; 对源程序进行编译连接; 可执行文件; 源程序; 伪指令是由编译器来执行的指令; 定义一个段segment和end; ends是段的结束; 汇编源程序=伪指令+汇编指令; 程序经编译连接后变为机器码; (2019-7-23)
23. Ep 21, 第一个程序02; 源程序; 程序返回; DOS中的程序运行; 段结束/程序结束/程序返回; 语法错误和逻辑错误; 编辑源程序; 上机; exe的执行; 以简化的方式进行编译和连接; 关于编译和连接; 每个源程序编译为目标文件后, 再用连接程序连接到一起, 生成一个可执行文件; 程序中调用了某个库文件中的子程序, 需要将这个库文件和该程序生成的目标文件连接到一起, 生成一个可执行文件; 对于连接的过程, 可执行文件是我们要得到的最终结果; (2019-7-24)
24. Ep 22, 第一个程序03; 可执行文件中的程序装入内存并运行的原理; 操作系统的外壳shell; 用户通过shell来操作计算机系统; command设置CPU的CSIP指向程序的第一条指令; edit, compile, link, command, cpu; 程序执行过程的跟踪; exe文件中的程序加载过程; (2019-7-24)
25. Ep 23, [BX]和loop指令01; [bx]和内存单元描述; loop; 描述性符号(); 约定符号idata表示常量; [bx]; 问题5.1分析; (2019-7-25)
26. Ep 24, [BX]和loop指令02; loop指令; CX中的值影响着LOOP指令的执行结果; 任务3, 编程计算2^12, 结果存放AX中; 标号; (2019-7-25)
27. Ep 25, [BX]和loop指令03; loop指令; 在CX中存放循环次数; loop指令中的标号所标识地址要在前面; 要执行的程序段, 要写在标号和loop指令之间; mov cx, loop times; s: code segment; loop s; 问题5.2; 在debug中跟踪用loop指令实现的循环程序; 在汇编源程序中, 数据不能以字母开头, 所以要在前面加0; (2019-7-25)
28. Ep 26, [BX]和loop指令04; 温故而知新; [BX]作为偏移地址和DS配合; loop和CX合作; debug和汇编编译器MASM对指令的不同处理; 对比汇编程序实现; 我还是要来啰嗦总结下; (2019-7-27)
29. Ep 27, [BX]和loop指令05; loop和[BX]的联合应用; 问题5.4; 分析; 不能用常量来给出内存单元的地址, 而应用变量; 段前缀; (2019-7-27)
30. Ep 28, [BX]和loop指令06; 一段安全的空间; 段前缀的使用; 程序代码分析; 改进的程序分析; 优化的策略在编程中非常重要; 没有小结的小结; (2019-7-27)
31. Ep 29, 包含多个段的程序01; 没有目录的目录; 在代码段中使用数据; dw = define word; end可以通知编译器程序的结束, 还可以通知编译器程序的入口; (2019-7-29)
32. Ep 30, 包含多个段的程序02; 在代码段中使用栈; 检测点6.1; 将数据代码栈放入不同的段; 代码段数据段栈段完全是我们的安排; (2019-8-7)
33. Ep 31,包含多个段的程序03; 实验五; (2019-8-7)
34. Ep 32, 更灵活定位内存地址01; 引言; and和or指令; and指令的一点功能; or指令; 关于ASCII码; 以字符形式给出数据; 大小写转换的问题; 中文符号是unicode; (2019-8-8)
35. Ep 33, 更灵活定位内存地址02; (bx) + idata; 用[bx+idata]的方式进行数组的处理; SI和DI; (2019-8-9)
36. Ep 34, 更灵活定位内存地址03; SI和DI; 问题7.2分析; 问题7.3分析; [bx+si]和[bx+di]; 问题7.4; 问题7.5; (2019-8-9)
37. Ep 35, 更灵活定位内存地址04; 不同的寻址方式的灵活应用; 问题7.6; 问题7.7; (2019-8-11)
38. Ep 36, 更灵活定位内存地址05; 问题7.8; 汇编语言的注释是分号; (2019-8-11)
39. Ep 37, 更灵活定位内存地址06; 问题7.9; 小小的小结; 寻址方式; 实验六; (2019-8-21)
40. Ep 38, 数据处理的两个基本问题01; 引言; bx, si, di, bp; right or wrong; 机器指令处理的数据所在位置; 读取, 写入, 运算; CPU内部, 内存, 端口; 汇编语言中数据位置的表达; 立即数; 寄存器; 段地址SA和偏移地址EA; 显性给出存放段地址的寄存器; 寻址方式; 寻址方式小结; (2019-8-21)
41. Ep 39, 数据处理的两个基本问题02; 寻址方式小结; 直接寻址; 寄存器间接寻址; 寄存器相对寻址; 基址变址寻址; 相对基址变址寻址; 指令要处理的数据有多长; 通过寄存器名指明要处理的数据的尺寸; 新方法; 寻址方式的综合应用; (2019-8-26)
42. Ep 40, 数据处理的两个基本问题03; C语言描述; 汇编语言描述; [bx+si+idata]的寻址方式为结构化数据的处理提供了方便; (2019-8-26)
43. Ep 41, 数据处理的两个基本问题04; div指令; 除数8位或16位; 被除数放在DX和AX中; div指令格式; div byte ptr [bx+si+8]; div word ptr [bx+si+8]; 利用除法指令计算100001/100; 编程实现1; 编程实现2; 伪指令dd; 问题8.1; 8.9, dup; dup示例; dup是一个非常有用的操作符; 小结; (2019-8-27)
44. Ep 42, 实验七, 寻址方式在结构化数据访问中的应用; 我们要考虑的几个问题; 初始化阶段; 每次循环要执行的任务; 存放年份; 计算人均收入并存放; (2019-8-27)
45. Ep 43, 转移指令的的原理01; 引言; 9.1, 操作符offset; 问题9.1分析; 9.2, jmp指令; 9.3, 依据位移进行转移的jmp指令; jmp short 标号, 实现的是段内短转移; 汇编指令与机器码的对应示例; jmp near ptr 标号; 实现段内近转移; (2019-8-29)
46. Ep 44, 转移指令的原理02; 9.4, 转移的目的地址在指令中的jmp指令; jmp far ptr 标号; 段间转移, 又称远转移; 程序9.3; 9.5, 转移地址在寄存器中的jmp指令; jmp 16位寄存器; 9.6, 转移地址在内存中的jmp指令; jmp word ptr 内存单元地址(段内转移); jmp dword ptr 内存单元地址(段间转移); (2019-8-29)
47. Ep 45, 转移指令的原理03; 9.7 jcxz指令; 有条件转移指令; jcxz 标号; 检测点9.2; 9.8, loop指令; 检测点9.3; (2019-9-1)
48. Ep 46, 转移指令的原理04; 9.9, 根据位移进行转移的意义; 方便了程序在内存段中的浮动装配; 9.10, 编译器对转移位移超界的检测; 实验八; 实验九; (2019-9-1)
49. Ep 47, 实验八实验九详细讲解; 实验八; 分析一个奇怪的程序; 实验九; 根据材料编程; (2019-9-4)
50. Ep 48, call和ret指令01; 目录; 引言; call和ret指令都是转移指令, 它们都修改IP, 或同事修改CS和IP; 它们经常被共同用来实现自程序的设计; 10.1, ret和retf; retf指令用栈中的数据, 修改IP的内容, 从而实现近转移; retf指令用栈中的数据, 修改CS和IP的内容, 从而实现远转移; 检测点10.1; 10.2, call指令; 将当前的IP或CSIP压入栈中, 转移jump; 10.3, 依据位移进行转移的call指令; 检测点10.2; (2019-9-6)
51. Ep 49, call和ret指令02; 10.4, 转移的目的地址在指令中的call指令; 检测点10.3; 10.5, 转移地址在寄存器中的call指令; call 16位寄存器; 检测点10.4; 10.6, 转移地址在内存中的call指令; call word/dword ptr 内存单元地址; 检测点10.5; 10.7, call和ret的配合使用; 问题探究, 问题10.1; 问题10.1分析; (2019-9-6)
52. ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
53. -