一堆指令的执行

大纲

Ch4 指令系统

- （一）指令系统的基本概念
- （二）指令格式
  - 1. 指令的基本格式
    - 操作码 —— 指明了动作
    - 操作数（可能有0~n个）
      - 寻址方式 —— 怎么找这个操作数？
      - 形式地址
        - 形式地址（需转成有效地址EA）
        - 指令运行时需转成有效地址EA
    - 根据指令长度是否固定不变，分为两类
      - 定长指令字 —— 每次PC+="1"，"1"固定
      - 变长指令字 —— 每次PC+="1"，"1"不固定
  - 2. 定长操作码指令格式 —— 操作码的位数固定不变
  - 3. 扩展操作码指令格式 —— 操作码的位数可能会改变
- （三）寻址方式
  - 1. 有效地址的概念 —— EA：最重要访问的地址
  - 2. 要寻找什么？
    - 数据寻址：找数据 —— 找到你要操作的数据
    - 指令寻址：找下一条要执行的指令 —— 影响PC的值
  - 3. 常见的寻址方式

- （四）数据的对齐和大小端存放方式
  - 数据的对齐
    - 按边界对齐
    - 不按边界对齐
  - 存放方式
    - 大端存储（更符合人类视角）
    - 小端存储
- （五）CISC 和 RISC 的基本概念 —— 从机器级代码中可以看到的显著区别
  - 指令字是否定长？
  - 除了 Load、Store 之外，是否还有其他可以指令可以访存？
- （六）高级语言程序与机器级代码之间的对应
  - 1. 编译器、汇编器、链接器的基本概念
  - 2. 选择结构语句的机器级表示
  - 3. 循环结构语句的机器级表示
  - 4. 过程（函数）调用对应的机器级表示

解题方法

先观察汇编语言，是x86还是MIPS？

- ①观察是否有注释 —— 通常来说，真题中x86汇编语言不会给太多注释（考研中默认大家懂 x86）
- ②观察指令长度是否固定
  - x86属于CISC，指令长度不固定
  - MIPS属于RISC，指令长度固定
- ③观察寄存器名
  - x86的寄存器名为 eax、ebx、ecx、edx...
  - MIPS 的寄存器名为 R[0]、R[1]、R[2]...

x86

先搞懂C语言

基于C语言的逻辑分析机器指令

- 有没有分支结构？ —— 观察有没有 jxxx 指令
- 有没有循环结构？
  - 观察有没有 jxxx 指令
  - 观察有没有 loopxxx 指令
- 有没有函数调用？
  - 观察有没有 call 指令
  - 观察有没有 ret 指令
- 需不需要访问函数调用参数？观察在汇编语言中如何访问调用参数？ —— 注：函数调用参数一般在 [ebp+8]、[ebp+12] 等位置
- 需不需要定义局部变量？观察在汇编语言中如何访问局部变量？ —— 注：局部变量的存储地址一般在 [ebp-4]、[ebp-8]、[ebp-12] 等位置

MIPS

先搞懂C语言

基于C语言的逻辑分析机器指令

- 有没有分支结构？
- 有没有循环结构？
- 有没有函数调用？

考不考流水线？

- 对指令进行分类 —— 五类指令，每一类指令的运行原理要搞清楚
- 安排指令流水线
- 分析数据冒险、控制冒险

基础知识

x86汇编指令（重点关注 intel 格式）

- 算数、逻辑运算类指令 —— 加、减、乘、除、左移、右移 等指令
- 分支结构（if/else）
  - 无条件转移（goto） —— jmp （无条件转移指令），类似于C语言的goto
  - 条件转移指令（if/else） —— ① cmp A，B （本质是 A-B，A-B 的标志位信息会存放到PSW中）
    - 无符号数减法关注 CF、ZF
      - A==B —— 需满足 ZF==1
      - A!=B —— 需满足 ZF==0
      - A>=B —— 需满足 CF==0
      - A<=B —— 需满足 CF==1 || ZF==1
      - A>B —— 需满足 CF==0 && ZF==0
      - A<B —— 需满足 CF==1
    - 有符号数减法关注 OF、SF、ZF
      - A==B —— 需满足 ZF==1
      - A!=B —— 需满足 ZF==0
      - A>=B —— 需满足SF==OF
      - A<=B —— 需满足SF!=OF || ZF==1
      - A>B —— 需满足SF==OF && ZF==0
      - A<B —— 需满足SF!=OF && ZF==0

    转移类指令，通常采用相对寻址，用补码表示偏移量。补码的值通常意味着PC要 往前/往后 跳多少个地址（注意：偏移量的单位可能是字节，也可能是指令字长）

    x86属于CISC，指令字长不固定，因此转移类指令中一定是以"字节"为单位描述PC的偏移量

    ② jxxx （转移指令）
- 循环结构（for while）
  - xxxloopxxxx <label> —— 原理同跳转指令
  - 条件跳转指令 jxxxx 也可以用于实现循环
- 函数调用/返回
  - 调用指令 —— call <label>
    - 当前PC值压栈保存，再修改PC的值，跳转到<label>
    - 与无条件转移指令jmp的区别 —— call 指令会将函数返回地址(当前的PC)入栈
  - 返回指令 —— ret —— 恢复程序计数器PC，返回原函数

MIPS 汇编指令

- ①算数、逻辑运算类指令
  - 加 —— OP：add
  - 减 —— OP：sub
  - 乘 —— OP：mul
  - 除 —— OP：div
  - 左移 —— OP：shl
  - 右移 —— OP：shr
- ②无条件转移指令
  - j <lable>
  - b <lable>

  转移类指令，通常采用相对寻址，用补码表示偏移量。补码的值通常意味着PC要 往前/往后 跳多少个地址（注意：偏移量的单位可能是字节，也可能是指令字长）

  MIPS属于RISC，指令字长固定，因此转移类指令中可能会以"指令字长"为单位描述PC的偏移量（参考 2013年真题）

  对比二者
- ③条件转移指令 —— bxxx —— b开头，意思是 branch
- ④Load指令 —— Load 寄存器，内存地址
- ⑤Store指令 —— Store 内存地址，寄存器

  唯二的会访问内存的指令

- 结合五段式指令流水线考察
  - 数据冒险 —— 什么指令可能导致数据冒险？
    - 前面的指令：写某个寄存器 —— WB，第五个段
    - 后面的指令：读（同一个）寄存器 —— ID，第二个段
    - 分析思路
      - 一条一条指令从前往后分析，如果一条指令写了某个寄存器，则观察与之相邻的3条指令中，有没有哪条指令需要读同一个寄存器
      - 当后面的指令和前面的指令中间，还有其他三条指令时，才不会发生数据冒险
  - 控制冒险 —— 什么指令可能导致控制冒险？ —— 会改变PC值的
    - 无条件转移（注：call、ret属于无条件转移）
      - EX，第三个段改变PC
      - 处理控制冒险的方法：停两个周期，再取下一条指令（IF段），就不会发生控制冒险
    - 有条件转移
      - M，第四个段改变PC
      - 处理控制冒险的方法：停三个周期，再取下一条指令（IF段），就不会发生控制冒险
      - 若未发生转移（不满足条件），则不会发生控制冒险

x86汇编语言，AT&T 和 intel 格式的区别

	AT&T 格式	Intel 格式
目的操作数d、源操作数s	op s, d 注：源操作数在左，目的操作数在右	op d, s 注：源操作数在右，目的操作数在左
寄存器的表示	mov %ebx, %eax 注：寄存器名之前必须加"%"	mov eax, ebx 注：直接写寄存器名即可
立即数的表示	mov $985, %eax 注：立即数之前必须加"$"	mov eax, 985 注：直接写数字即可
主存地址的表示	mov %eax ,(af996h) 注：用 "小括号"	mov [af996h], eax 注：用 "中括号"

读写长度的表示	movb $5, (af996h)
	movw $5, (af996h)
	movl $5, (af996h)
	addb $4, (af996h)
	注：指令后加 b、w、l 分别表示读写长度为 byte、word、dword
		mov byte ptr [af996h], 5
		mov word ptr [af996h], 5
		mov dword ptr [af996h], 5
		add byte ptr [af996h], 4
		注：在主存地址前说明读写长度byte、word、dword

主存地址偏移量的表示	movl -8(%ebx), %eax *注：偏移量(基址)* movl 4(%ebx, %ecx, 32), %eax *注：偏移量(基址, 变址, 比例因子)*	mov eax, [ebx - 8] *注：[基址+偏移量]* mov eax, [ebx + ecx*32 + 4] *注：[基址+变址*比例因子+偏移量]*