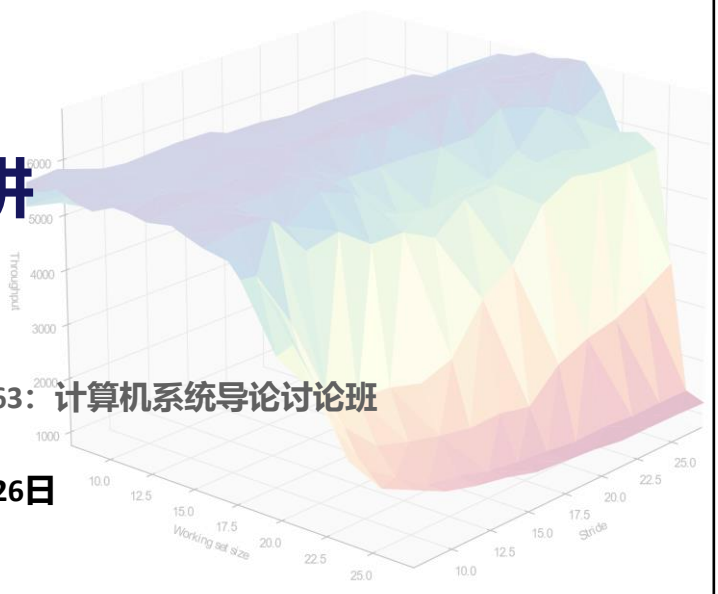
第四讲



PKU 04832363：计算机系统导论讨论班

王畅

2021年10月26日

记忆量比较大

- 通过先理解来帮助记忆，不要死记硬背
- 尝试和同学讨论，相互复述流程
- 通过做题确保重要的部分没有问题

Activity 1

轮流回答问题

设计自己的计算机——基本框架

- **对计算机的基本结构进行抽象**
  - Y86-64中的例子：内存、ZF/SF/OF等条件码、寄存器堆、机器状态（AOK、HLT、ADR、INS）、程序计数器
- **设计一套指令（助记码+具体编码）**
  - Y86-64中的例子：`irmovq V rB`，对应于计算机的可读编码是`0x30f[rB][V]`
- **设计硬件去读取、分析和执行这些指令**
  - 对应于5阶段模型以及其中的各种变化，详细写出之
  - 为什么要使用一些中间变量？（例如valA、valB）它们对应于一些有用的中间信号
- **硬件描述语言**
  - 用类似高级语言的方式写出信号应当遵循的逻辑和时序
  - 有现代软件能够通过这种描述语言直接生成优化的电路布局
    （这就是为什么这章会写一些看起来很不明觉厉的代码）

4

在进入细节之前，一定要先理解整个ISA章节叙述的逻辑是怎样的，整个Y86-64体系结构的设计流程是什么，有什么主要内容，为什么要引入这些内容。此外，注意识记各种符号和记号。

Question 1

- **首先，回忆rrmovq、irmovq、rmmovq、mrmovq、OPq、jXX、cmovXX、call、ret、pushq、popq指令的基本编码是怎样的？例如：rrmovq是0x20[rA][rB]，2字节。**
- **打开课本第246~247页，参照其中给出详细指令结构，完成下面5段代码的译码工作（可能是多条指令或者有错，冒号前面是起始地址）：**

1. 0x100:30f3fcffffffffffffff4063000800000000000
2. 0x200a06f800c020000000000000030f30a000000000000
3. 0x300:50540700000000000010f0b01f
4. 0x400:6113730004000000000000
5. 0x500:6362a0f0

主要是理解计算机实际读的是字节码，后面的译码过程就是要指导计算机机械地进行这一过程！答案如下：

1、irmovq $-4, %rbx、rmmovq %rsi, 0x800(%rbx)、halt

2、pushq %rsi、call proc、halt、proc: irmovq $10, %rbx、ret

3、mrmovq 7(%rsp), %rbp、nop、有错误、popq %rcx

4、loop: subq %rcx, %rbx、je loop、halt

5、xorq %rsi, %rdx、有错误

要注意，立即数是按字节反序填充的（小端法）！

顺序实现的5阶段

- 熟记课本264页各种notation的含义和各阶段的作用！
- 取指：取出的内容有icode（什么指令）、ifun（子功能）、rA、rB、valC（根据指令类型推断是否需要）；计算的内容：下一条指令的地址valP
- 译码：读寄存器（rA、rB对应于valA、valB，也有可能读rsp，存入valB）
- 执行：主要涉及ALU，即需要算术运算的指令（包括加减栈指针、检查传送条件等），计算结果为valE，可能会设置条件码
- 访存：读写内存，读出的那个值为valM
- 写回：把结果写回到寄存器文件中
- 更新PC

6

这里的notation不能弄混，对做题有格外的重要性；可以通过缩写来辅助记忆。这里顺序执行的逻辑是从上至下，当然，其实是可以并行操作的。

表格填写总结（表4-18~4-21，理解+记忆）

阶段	操作
取指	确定Icode:ifun、rA:rB、valC 计算valP=PC+?
译码	计算valA、valB：来自rA、rB、或%rsp
执行	可能运算：valE=valB OP valA / 0+valA / 0+valC / valB+valC / valB+(±8) Set CC 条件指令：Cnd=Cond(CC, ifun)
访存	写内存：M8[valE]=valA / valP 或者读内存：valM=M8[valE] / M8[valA]
写回	R[%rsp]=valE R[rA]=valM、R[rB]=valE if(Cnd)R[rB]=valE
更新PC	PC=valP / valC / valM Cnd?valC:valP

7

注意这里访存的格式：读X个字节以及地址都要指出。valC的来源是立即数（包括绝对内存地址、内存偏移量和运算中的立即数几种可能）。

理论上说，valA和valB的地位是平等的，但如果指令只使用一个寄存器，那么一般会使用rB，所以不存在valA+valC这种可能。执行过程中，依次产生这种操作的可能指令为算术运算、rrmovq/cmovXX、irmovq、rmmovq/mrmovq、push/pop/call/ret。

CC是机器中的条件码（运算产生），这里Cnd的设置格式要记住，表示通过CC和指令中的XX来确定Cnd为何。

访存过程中，写内存的两种可能分别是rmmovq/push、call；读内存的两种可能分别是mrmovq/pop。

写回过程中，写%rsp寄存器的可能是push/pop，写rA和rB的分别可能是pop/mrmovq、算术指令，最后要判断条件码的是cmovXX指令。

更新PC，valP对用的是一般指令，valC是无条件跳转或者call，valM是ret指令。

一些记忆trick

- 控制指令在取指阶段不取寄存器
- 只有OP操作会设置条件码
- 在译码阶段取%rsp的有pop push call ret，其中pop和ret都是将valA valB全部赋值为%rsp，目的是操作更整齐（雾），push和call则是将valB赋值为%rsp

例子

阶段	pushq rA
取指（分开各个部分）	icode:ifun <- M1[PC] rA:rB <- M1[PC+1] valP <- PC+2
译码（读寄存器）	valA <- R[rA] valB <- R[%rsp]
执行（算术运算+条件码）	valE <- valB+(-8)
访存	M8[valE] <- valA
写回	R[%rsp] <- valE
更新PC	PC <- valP

例子

阶段	cmovXX rA, rB
取指（分开各个部分）	icode:ifun <- M1[PC] rA:rB <- M1[PC+1] valP <- PC+2
译码（读寄存器）	valA <- R[rA]
执行（算术运算+条件码）	valE <- 0+valA **Cnd <- Cond(CC, ifun)**
访存	**none**
写回	if (Cnd) R[rB] <- valE
更新PC	PC <- valP

考点2：顺序执行的5阶段

- 选择题考法为回忆课本内容
- 大题一般考法是新设计一个指令，要求分析5个阶段的对应逻辑
- 熟练掌握notation、对5阶段的基本工作和基本指令的对应倒背如流（雾）是关键!
- Section 7 练习2、3
- Section 8 练习1、2、4、8、12、16
- Question 2：练习选讲

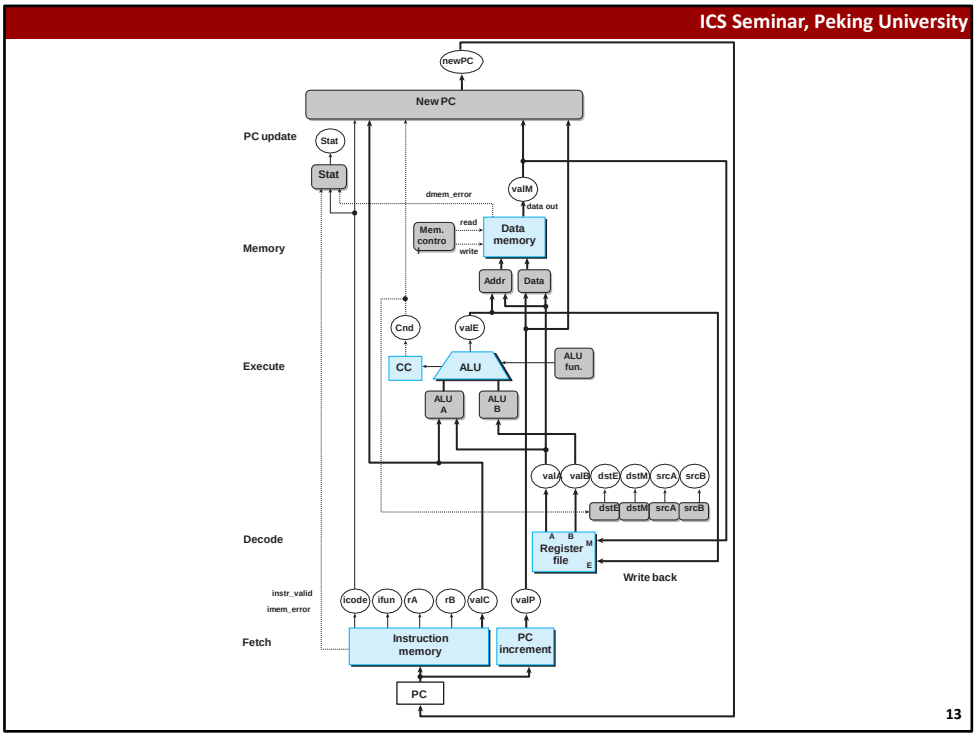
思考方法：先决定这个阶段要产生哪些信号，要用到哪些信号，再回忆或推断这条指令是否需要产生/用到这些信号。不要遗漏细节。

SEQ的硬件实现

- **这张图非常重要，是之后理解PIPE的基础，特别要熟悉其中的notation！（整张图应该是记不住的）**
- **srcA和srcB分别是两个寄存器的源；dstE是valE的目标寄存器，dstM是valM的目标寄存器（这四个在PIPE中还会用到）**
- **结合课本4.3.4节的阅读（这一节的实现也要大体记下来）**

12

结合HCL实现来理解这张图中的各种变量的含义。去读一下instr_valid、need_regids、need_valC、srcA、srcB、dstE、dstM、aluA、aluB、set_cc、mem_addr、mem_data、mem_read、mem_write、stat、new_pc等的设置逻辑。一般不会要求大家完整写出，但是可以填空。



图中的寄存器文件有两个写端口，M和E。当同一个周期内两个端口试图同时对一个寄存器进行写时，只有优先级较高的M端口会发生写（这个问题主要出现在popq %rsp这条指令中）。

灰色：控制逻辑（是组合电路）；白色方框：时序逻辑（是时序电路）。大体上讲，把SEQ处理器改成PIPE之后，流水线的各阶段引入的寄存器就是用来保存这些阶段的状态的，因此记住每个阶段用到的状态变量很重要。

有一些HCL逻辑或者信号逻辑可能不符合直觉，这里都要参考硬件实现来理解（必须符合硬件实现）。例如，访存阶段如果要用valA、valB，即便是二者地位相同，也只能用valA，因为只有它传向访存阶段。又如，不存在valA+valC这样的执行过程，这从图中的信号流向中也能看出（这也是为什么单个寄存器一般用的rB）。再如，jXX指令不需要计算0+valC，而irmovq型指令却需要，是因为valC可以直接参与new PC的运算逻辑，但写回阶段只能由valE参与。

理解时序（了解）

- **组合逻辑和时序逻辑**
  - 组合逻辑电路在逻辑功能上的特点是任意时刻的输出仅仅取决于该时刻的输入，与电路原来的状态无关。而时序逻辑电路在逻辑功能上的特点是任意时刻的输出不仅取决于当时的输入信号，而且还取决于电路原来的状态，或者说，还与以前的输入有关
  - 举例：ALU属于组合逻辑，因为输入变化之后它会在一定的延时下立刻作出响应；PC属于时序逻辑，因为它是受时钟控制的，每次时钟上升（或下降）沿才会响应输入变化；寄存器文件也是时序逻辑，因为一个周期更新一次
  - 比喻：组合逻辑是照相机的取景器，时序逻辑虽然也能看到取景器，但它是快门+照相机的照片预览
- **需要通过设计时序，同步各时序电路的行为**
  - 从不回读原则

14

这里更重要的是理解状态的更新行为，凡是由时序逻辑控制的状态，例如状态码、程序计数器、寄存器文件、数据内存都是只在下一个周期来临的时候得到实际的更改（而组合逻辑，例如ALU的输出信号都是随时更改的）。这对流水线处理器的理解也至关重要。

Activity 2

问题求解

考点1：RISC和CISC的区别

- 通读课本249页表格，确保每个举的例子都熟悉
- Section 7 练习1
- Section 8 练习5、9、11、14

考点3：流水线的原理与计算

- **熟练掌握流水线的pros和cons**
  - 提高吞吐率、增加单条指令执行时间、会发生冒险、级数不能太深、划分要尽量均衡、性能提升有上限等
- **会设计流水线，计算延时和吞吐率**
  - 吞吐率是延时的倒数
  - 顺序执行：延时由延时最长的通路决定
  - 流水线化：延时由延时最长的寄存器间时间加上寄存器的延时计算
  - 设计流水线：尽量均匀划分，有多条通路的每条都要划分
- **流水线指令运算时间的计算**
  - 通过画简单时序图来分析
- **Section 7 练习4、5、6**
- **Section 8 练习3、6、7、10、11、13、15**

any questions?

Thanks & 感谢观看