Processor Arch: ISA&Logic;Sequential

TA: 朱睿冬 王非石

2021.10.26

目录

- Y86-64指令集
- 逻辑设计
- Y86-64的顺序实现

Y86-64指令集

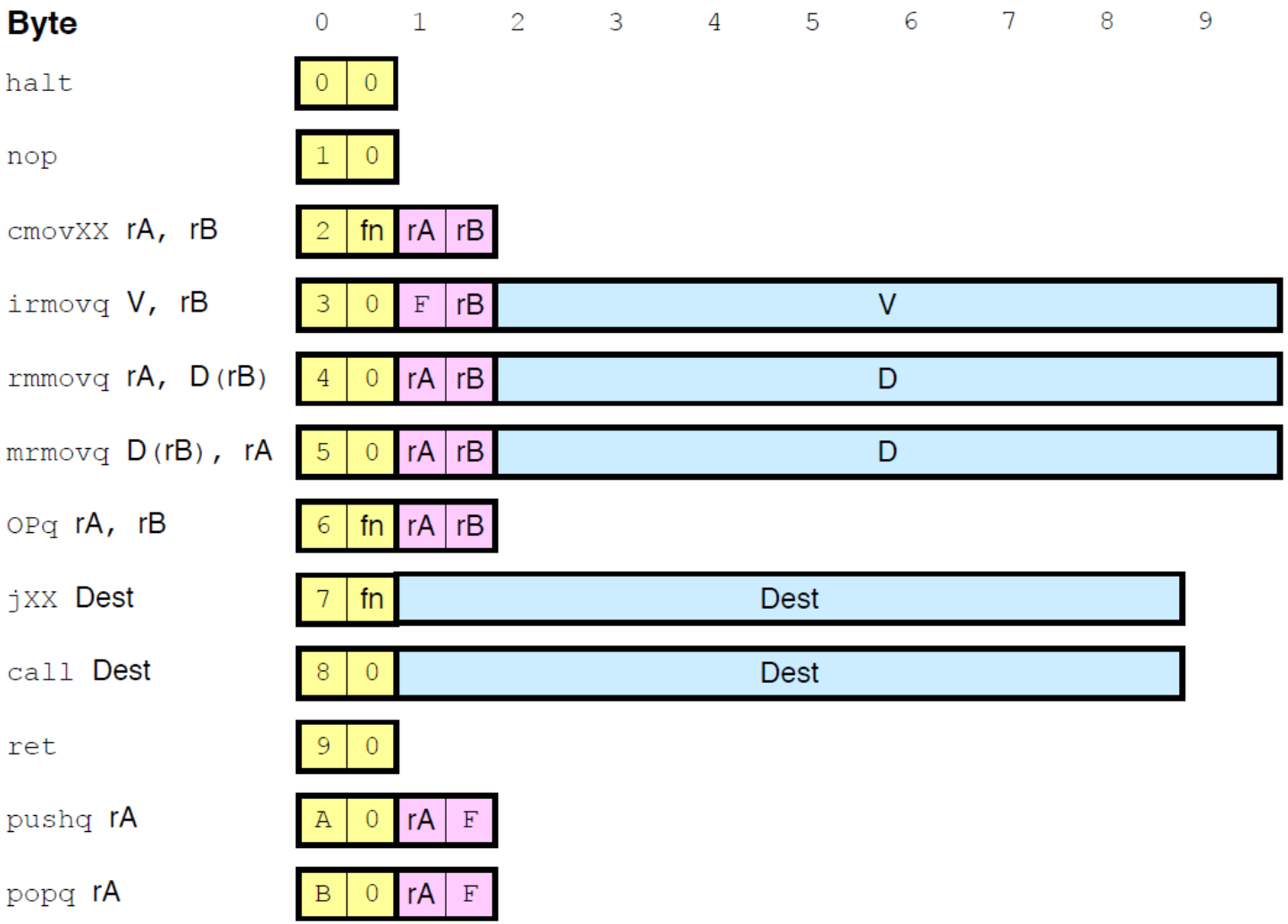
- 程序员可见状态
  - 程序寄存器（15个，64位）
    - 省略%r15以简化指令的编码
  - 条件码（ZF,SF,OF）
  - 程序计数器（PC）
  - 内存（虚拟地址）
  - 状态码（Stat）：表明程序执行的总体状态

Y86-64指令集

- 指令与指令编码
  - 指令功能码fn

Operations			Branches				Moves			
addq	6	0	jmp	7	0	jne	7	4	rrmovq	2	0	cmovne	2	4
subq	6	1	jle	7	1	jge	7	5	cmovle	2	1	cmovge	2	5
andq	6	2	jl	7	2	jg	7	6	cmovl	2	2	cmovg	2	6
xorq	6	3	je	7	3				cmove	2	3			

- 内存访问：基址+偏移量
  - 无变址，无伸缩
  - 不允许立即数到内存
- 绝对地址寻址
- 整数采用小端法编码



Y86-64指令集

- Y86-64程序
  - 与x86-64的汇编代码比较
    - Y86-64需要将常数加载到寄存器
    - 从内存中读取数值并与一个寄存器相加，Y86-64需要两条指令，x86-64只需要一条
  - 完整的Y86-64程序
    - 需要汇编器伪指令
    - 需要程序员手动执行编译器、链接器和运行时系统所需要完成的任务
    - YAS：汇编器；YIS：指令集模拟器

x86-64 code

```
    long sum(long *start, long count)
    start in %rdi, count in %rsi
1   sum:
2     movl    $0, %eax          sum = 0
3     jmp     .L2               Goto test
4   .L3:                      loop:
5     addq    (%rdi), %rax      Add *start to sum
6     addq    $8, %rdi          start++
7     subq    $1, %rsi          count--
8   .L2:                      test:
9     testq   %rsi, %rsi        Test sum
10    jne     .L3               If !=0, goto loop
11    rep; ret                  Return
```

Y86-64 code

```
    long sum(long *start, long count)
    start in %rdi, count in %rsi
1   sum:
2     irmovq $8,%r8           Constant 8
3     irmovq $1,%r9           Constant 1
4     xorq %rax,%rax          sum = 0
5     andq %rsi,%rsi          Set CC
6     jmp     test            Goto test
7   loop:
8     mrmovq (%rdi),%r10      Get *start
9     addq %r10,%rax          Add to sum
10    addq %r8,%rdi           start++
11    subq %r9,%rsi           count--.  Set CC
12  test:
13    jne    loop             Stop when 0
14    ret                     Return
```

Y86-64指令集

- RISC和CISC
  - 教材249页表格
  - 举例
    - CISC: x86
    - RISC: ARM, RISC-V, MIPS

11、关于 RISC 和 CISC 的描述，正确的是：

A. CISC 指令系统的指令编码可以很短，例如最短的指令可能只有一个字节，因此 CISC 的取指部件设计会比 RISC 更为简单。

B. CISC 指令系统中的指令数目较多，因此程序代码通常会比较长；而 RISC 指令系统中通常指令数目较少，因此程序代码通常会比较短。

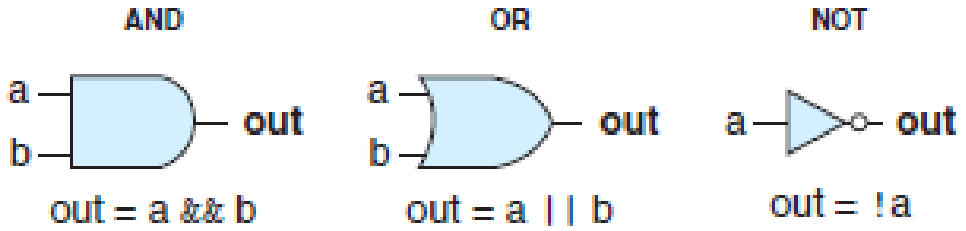
C. CISC 指令系统支持的寻址方式较多，RISC 指令系统支持的寻址方式较少，因此用 CISC 在程序中实现访存的功能更容易。

D. CISC 机器中的寄存器数目较少，函数参数必须通过栈来进行传递；RISC 机器中的寄存器数目较多，只需要通过寄存器来传递参数。

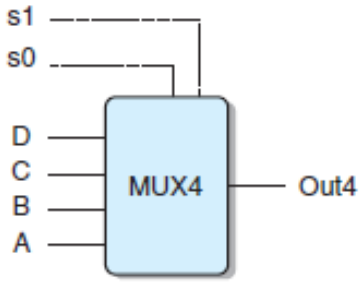
答案：C。
A：CISC取值部件设计更复杂
B：CISC一般程序代码更短
D：RISC也可能通过栈传递参数

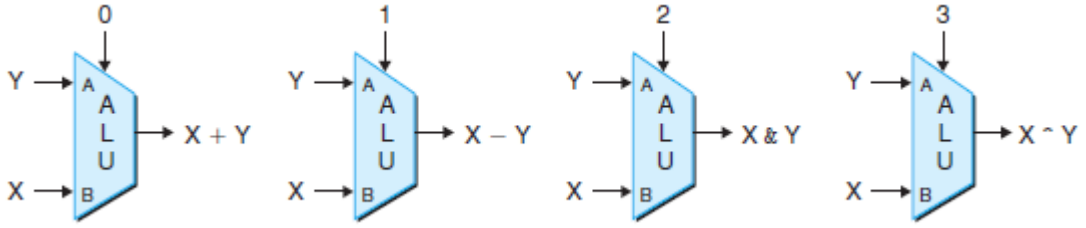
逻辑设计

- 逻辑门



- 组合电路
  - 逻辑门组合成的网
  - 构建存在限制
    - 输入；输出；整体无环
  - 输入到输出的一个函数
    - 描述方式：HCL
  - 常见部件
    - 多路复用器
    - ALU



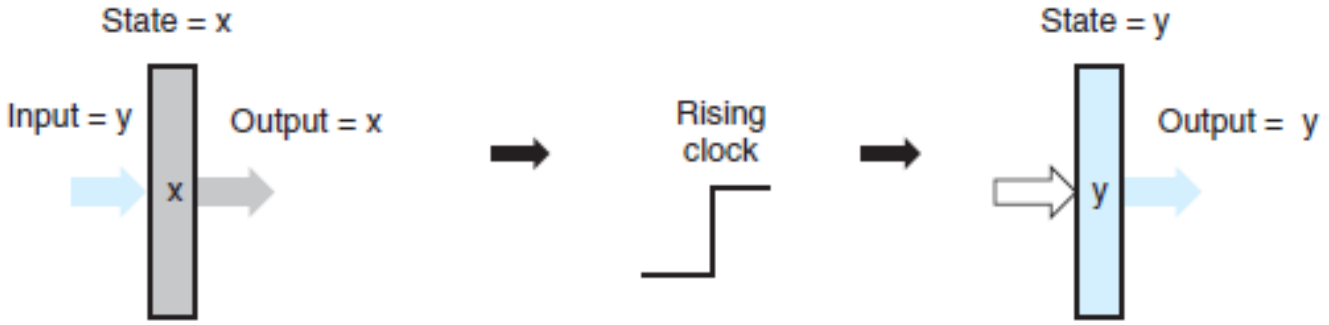


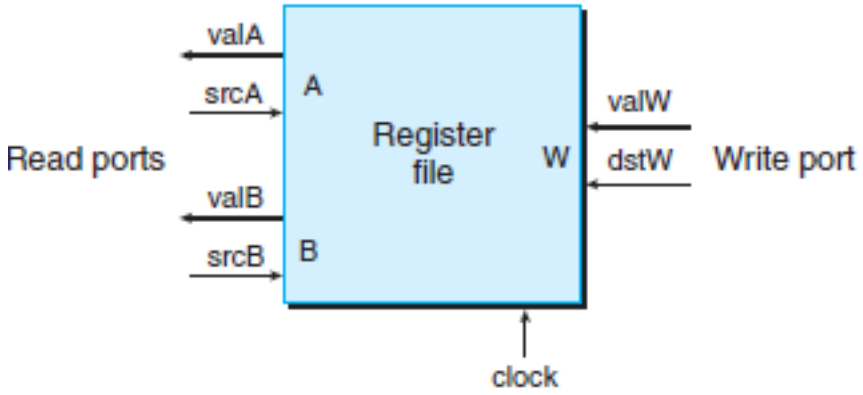
逻辑设计

- HCL
  - 数据类型：bool, int
  - 操作
    - 简单的布尔表达式:　bool out = (s&&a)||(!s&&b)
    - 情况表达式（多选器） word out = [select1 : expr1, select2 : expr2 ...]
    - 集合关系 bool s = code in { item1, item2, ... }
  - 与C语言逻辑运算的区别
    - 组合电路输出持续响应输入的变化
    - HCL无C语言中的“短路”，不会部分求值

逻辑设计

- 存储器和时钟
  - 时序电路
    - 输出是输入和当前状态的函数
  - 两类存储设备
    - 时钟寄存器
      - 时钟上升沿到来时输入信号被加载到寄存器并输出
      - 和机器级编程中的“寄存器”的区别
    - 随机访问存储器
      - 内存系统
      - 寄存器文件
      - 读类似组合逻辑，写受时钟控制





Y86-64的顺序实现

- 阶段化
  - 取指：从内存读取指令字节、同时增加PC
  - 译码：从寄存器文件中读入最多两个操作数
  - 执行：ALU进行指令指明的操作或对栈指针进行增减，同时可能设置条件码
  - 访存：将数据写入内存或读出数据
  - 写回：将结果写回到寄存器文件
  - 更新PC：将PC设置成下一条指令的地址

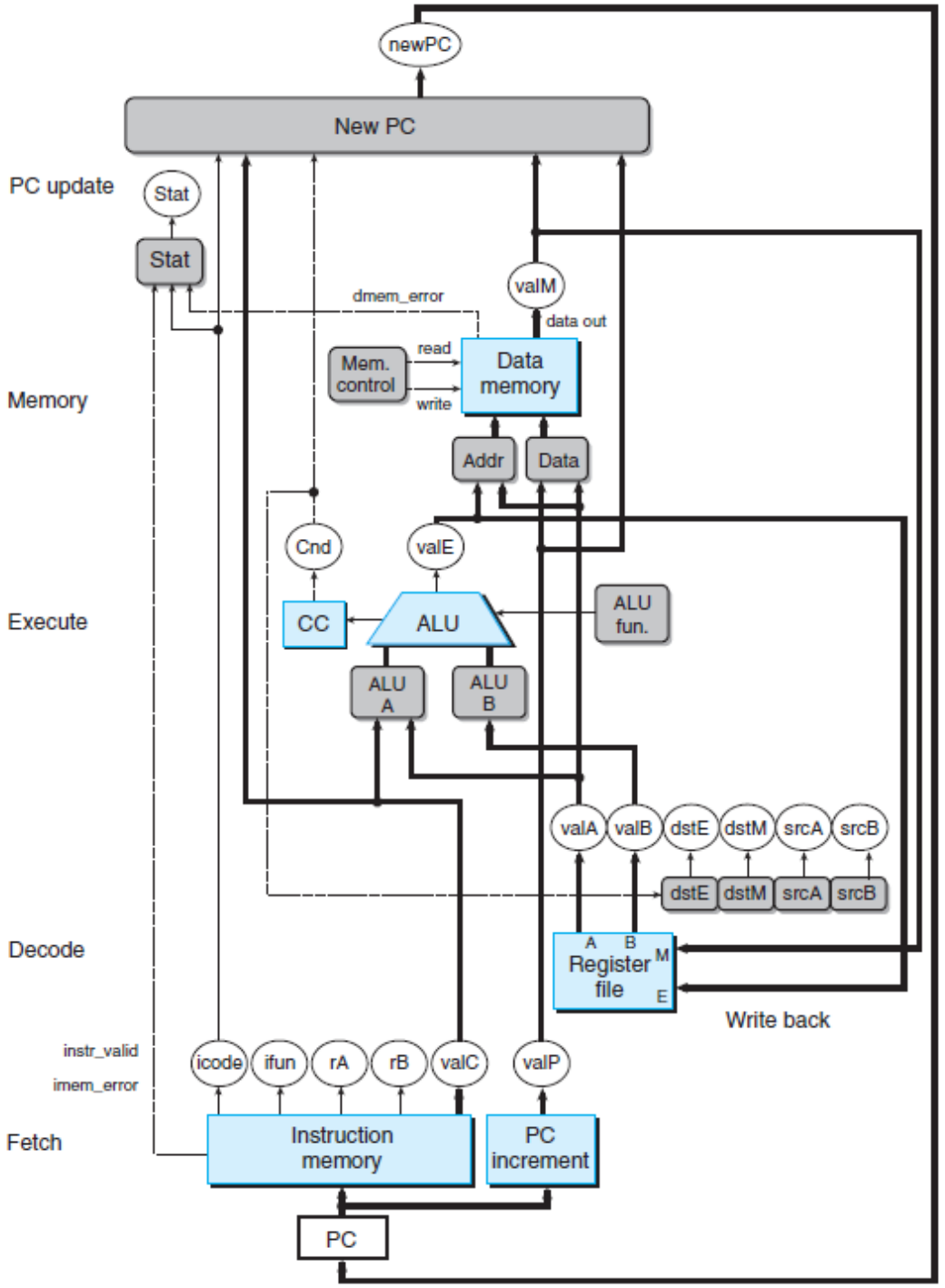
Y86-64的顺序实现

- 操作示例
  - 教材4.3中所有类似表格都要记下来

Stage	`OPq` rA, rB	`rrmovq` rA, rB	`irmovq` V, rB
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC + 1] valP ← PC + 2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC + 1] valP ← PC + 2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC + 1] valC ← M_8[PC + 2] valP ← PC + 10
Decode	valA ← R[rA] valB ← R[rB]	valA ← R[rA]	
Execute	valE ← valB OP valA Set CC	valE ← 0 + valA	valE ← 0 + valC
Memory			
Write back	R[rB] ← valE	R[rB] ← valE	R[rB] ← valE
PC update	PC ← valP	PC ← valP	PC ← valP

Y86-64的顺序实现

- SEQ硬件结构
  - 从抽象视图到硬件结构
  - 抽象视图
    - 关注硬件和各处理阶段的关联
    - 省略控制逻辑块
  - 硬件结构
    - 增加控制逻辑和线路类型
    - 明确信号在硬件之间的传播路径

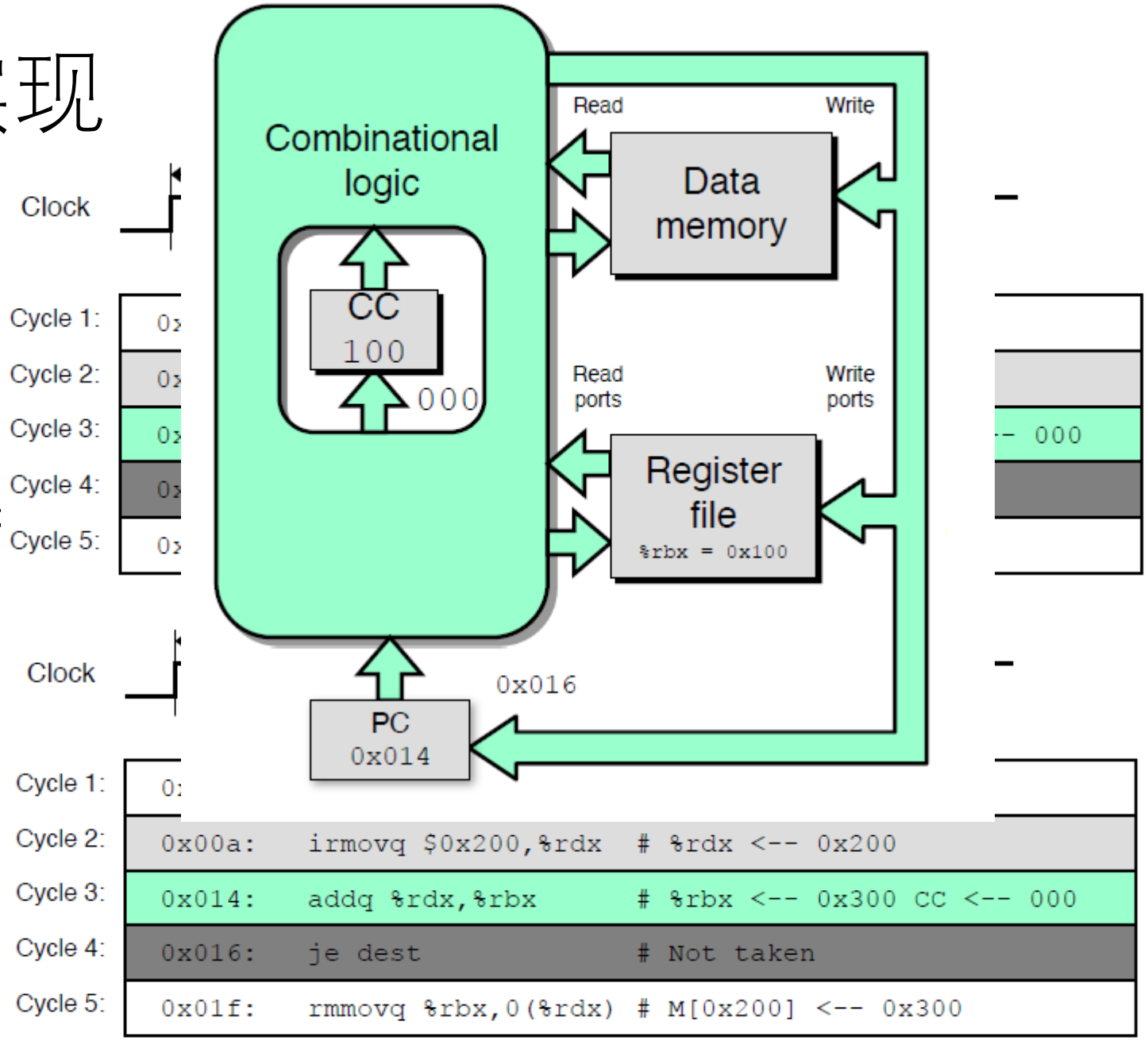


Y86-64的顺序实现

- SEQ的时序
  - SEQ是如何保证硬件结构可以实现顺序操作的
    - 组合逻辑无需考虑（包括随机访问存储器的读）
    - 需要考虑时序的硬件单元：PC，条件码寄存器，数据内存和寄存器文件
  - 原则：从不回读（处理器从来不需要为了完成一条指令的执行而去读由该指令更新了的状态）
    - 例：没有指令既设置又读取条件码

Y86-64的顺序实现

- SEQ的时序
  - 周期3刚开始时
    - 状态单元读入第二条指令 irmovq所赋予的状态
    - 组合逻辑则开始对第三条指令更新做出反应
  - 周期3即将结束时
    - 状态单元仍保持第二条指令 irmovq所赋予的状态
    - 组合逻辑已经生成了addq指令对应的值



Y86-64的顺序实现

- SEQ阶段的实现
  - 重点：控制逻辑块的HCL描述
  - 需要全记住（但无需死记硬背）
    - 结合指令功能和硬件结构记忆