嵌入式系统安全与设计

庄连生

Email: { *lszhuang@ustc.edu.cn* }

***Fall 2019*，USTC**

University of Science and Technology of China

第2讲 嵌入式硬件基础

内容提要：

- CISC指令集和RISC指令集
- 冯·诺依曼体系结构和哈佛体系结构
- 桶型移位器
- 流水线

……

嵌入式系统

嵌入式系统软件部分

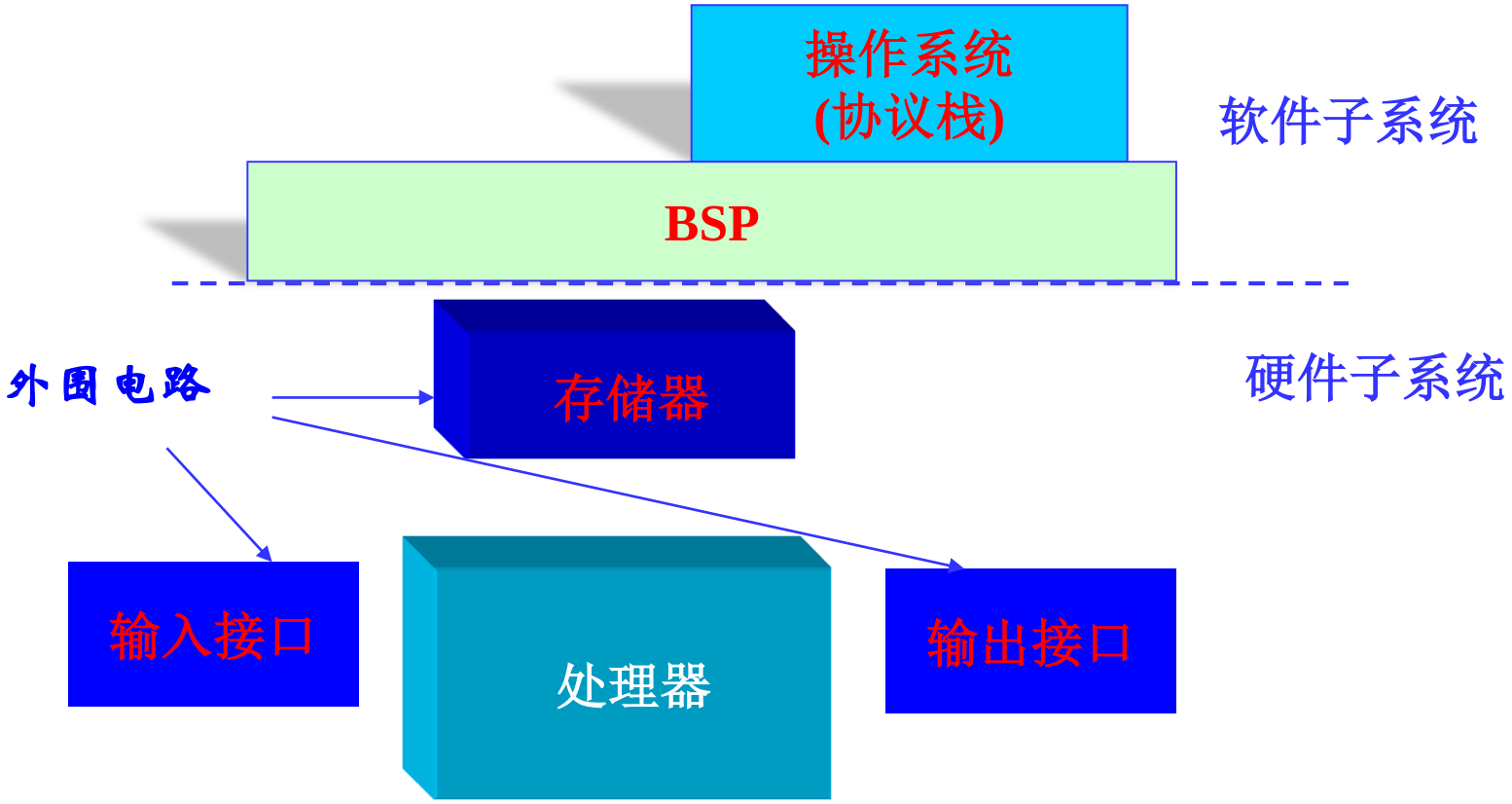
如人的大脑，决定了硬件的操作模式。通过良好的操作系统以及应用程序，把硬件功能发挥到极至。

嵌入式系统硬件部分

如人的手、脚、神经等部位，决定了嵌入式系统的先天功能。如运算能力和I/O接口等。

嵌入式系统

- 嵌入式系统组成



CISC和RISC

CISC：复杂指令集（Complex Instruction Set Computer）

具有大量的指令和寻址方式，指令长度可变

8/2原则：80%的程序只使用20%的指令

大多数程序只使用少量的指令就能够运行。

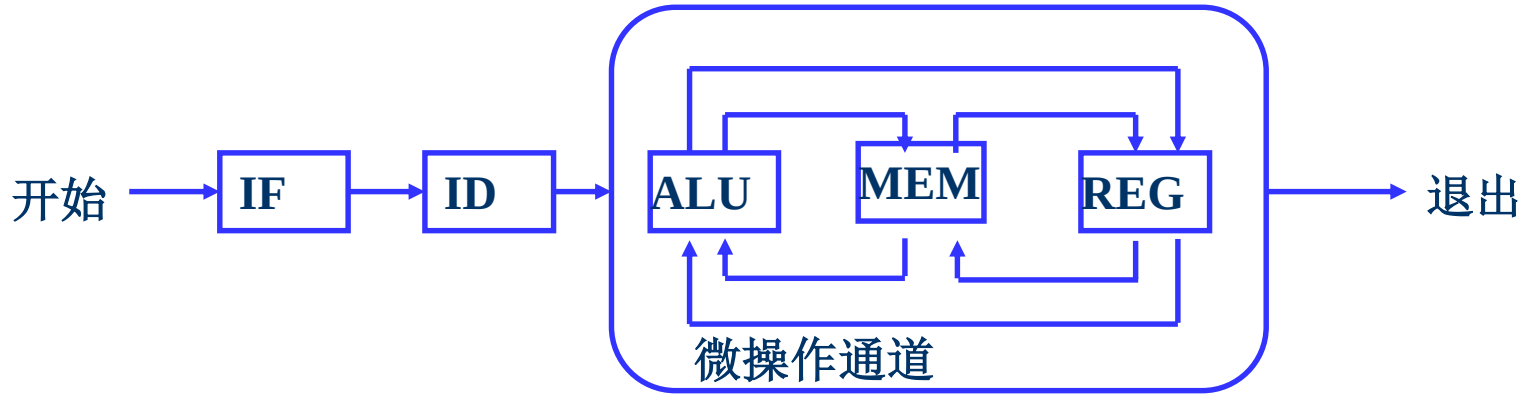
RISC：精简指令集（Reduced Instruction Set Computer）

只包含最有用的指令，指令长度固定

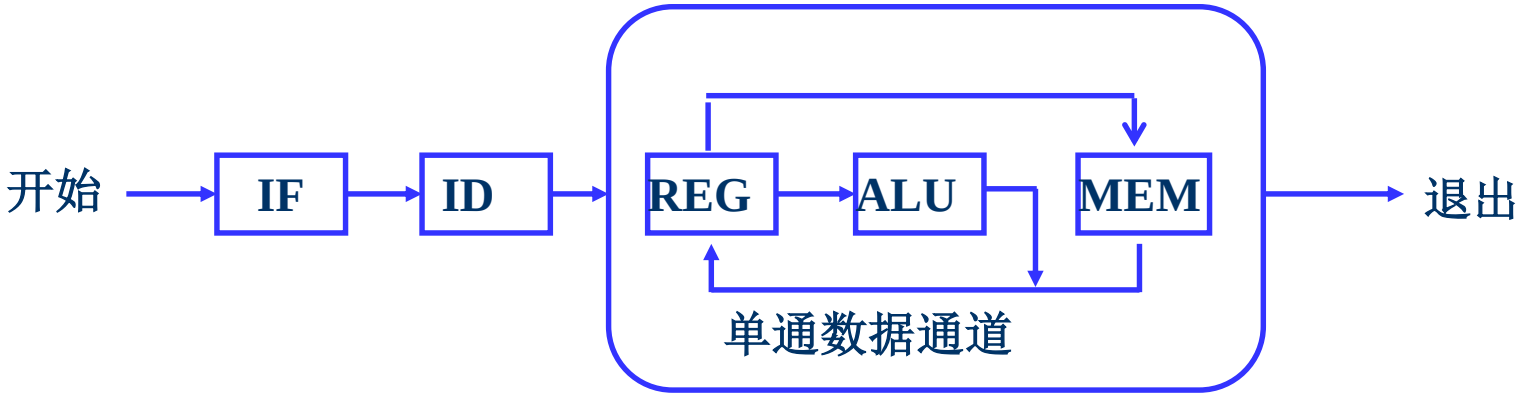
确保数据通道快速执行每一条指令

使CPU硬件结构设计变得更为简单

CISC与RISC的数据通道



CISC：寻址方式复杂



RISC：Load/Store结构

CISC的背景和特点

- 增强指令功能，设置一些功能复杂的指令，把一些原来由软件实现的、常用的功能改用硬件的（微程序）指令系统来实现
- 为节省存储空间，强调高代码密度，指令格式不固定，指令可长可短，操作数可多可少
- 寻址方式复杂多样，操作数可来自寄存器，也可来自存储器
- 采用微程序控制，执行每条指令均需完成一个微指令序列（微程序）
- CPI > 5，指令越复杂，CPI越大。

CISC的主要缺点

- **指令使用频度不均衡**
  - ✓ 高频度使用的指令占据了绝大部分的执行时间，扩充的复杂指令往往是低频度指令。
- **大量复杂指令的控制逻辑不规整，不适于VLSI工艺**
  - ✓ VLSI的出现，使单芯片处理机希望采用规整的硬联逻辑实现，而不希望用微程序，因为微程序的使用反而制约了速度提高。(微码的存控速度比CPU慢5-10倍)。
- **软硬功能分配**
  - ✓ 复杂指令增加硬件的复杂度，使指令执行周期大大加长，直接访存次数增多，降低了CPU性能。
- **不利于先进指令级并行技术的采用**
  - ✓ 流水线技术

RISC基本设计思想

- 减小CPI: CPUtime=Instr_Count * CPI * Clock_cycle
- 精简指令集：保留最基本的,去掉复杂、使用频度不高的指令
- 采用Load/Store结构，有助于减少指令格式，统一存储器访问方式
- 采用硬接线控制代替微程序控制

RISC:减少指令平均执行周期数

- **CPUtime= Instr_Count *CPI * Clock_cycle**
  - ✓ $IC_{RISC} > IC_{CISC}$， 40%
  - ✓ $CC_{RISC} \sim CC_{CISC}$，大致相当
  - ✓ $CPI_{RISC} < CPI_{CISC}$， 30%
  - ✓ 超标量、流水线等系统结构， 目标在于减小CPI， 可使CPI<1

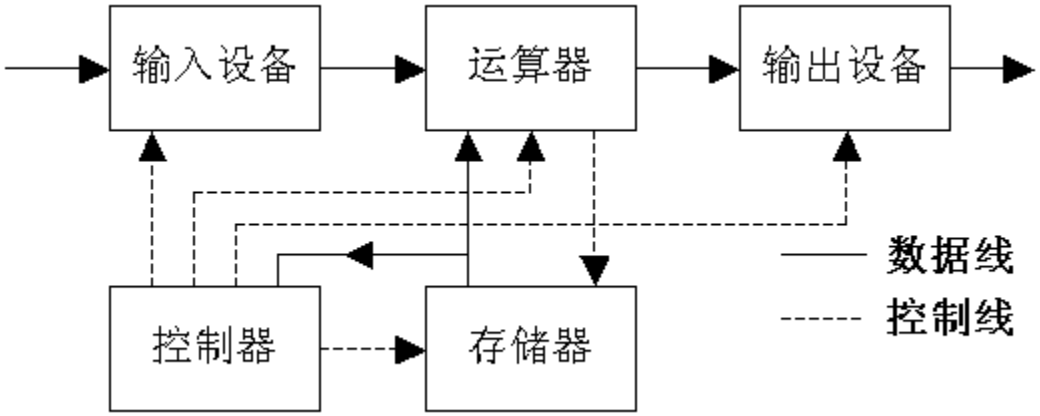
典型的高性能RISC处理器

- SUN公司的SPARC(1987)
- MIPS公司的MIPS(1986)
- IBM, Motorola公司的PowerPC
- DEC、Compac公司的Alpha AXP
- HP公司的PA-RISC

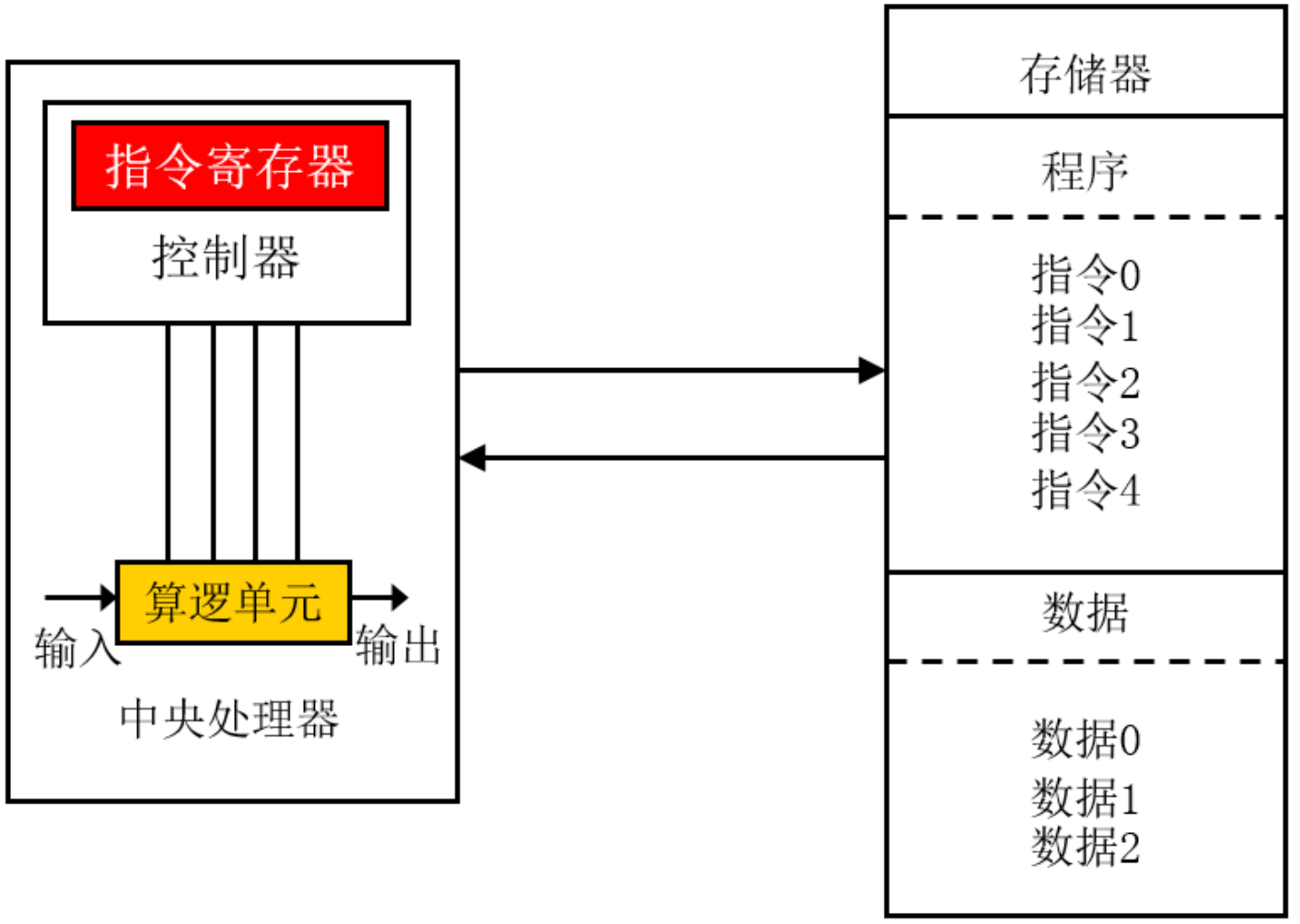
CISC与RISC的对比

类别	CISC	RISC
指令系统	指令数量很多，通常>200	较少，通常少于100
执行时间	有些指令执行时间很长，如整块的存储器内容拷贝；或将多个寄存器的内容拷贝到存贮器	没有较长执行时间的指令
编码长度	编码长度可变，1-15字节	编码长度固定，通常为4个字节
寻址方式	寻址方式多样	简单寻址
操作	可以对存储器和寄存器进行算术和逻辑操作	只能对寄存器进行算术和逻辑操作，Load/Store体系结构
编译	难以用优化编译器生成高效的目标代码程序	采用优化编译技术，生成高效的目标代码程序

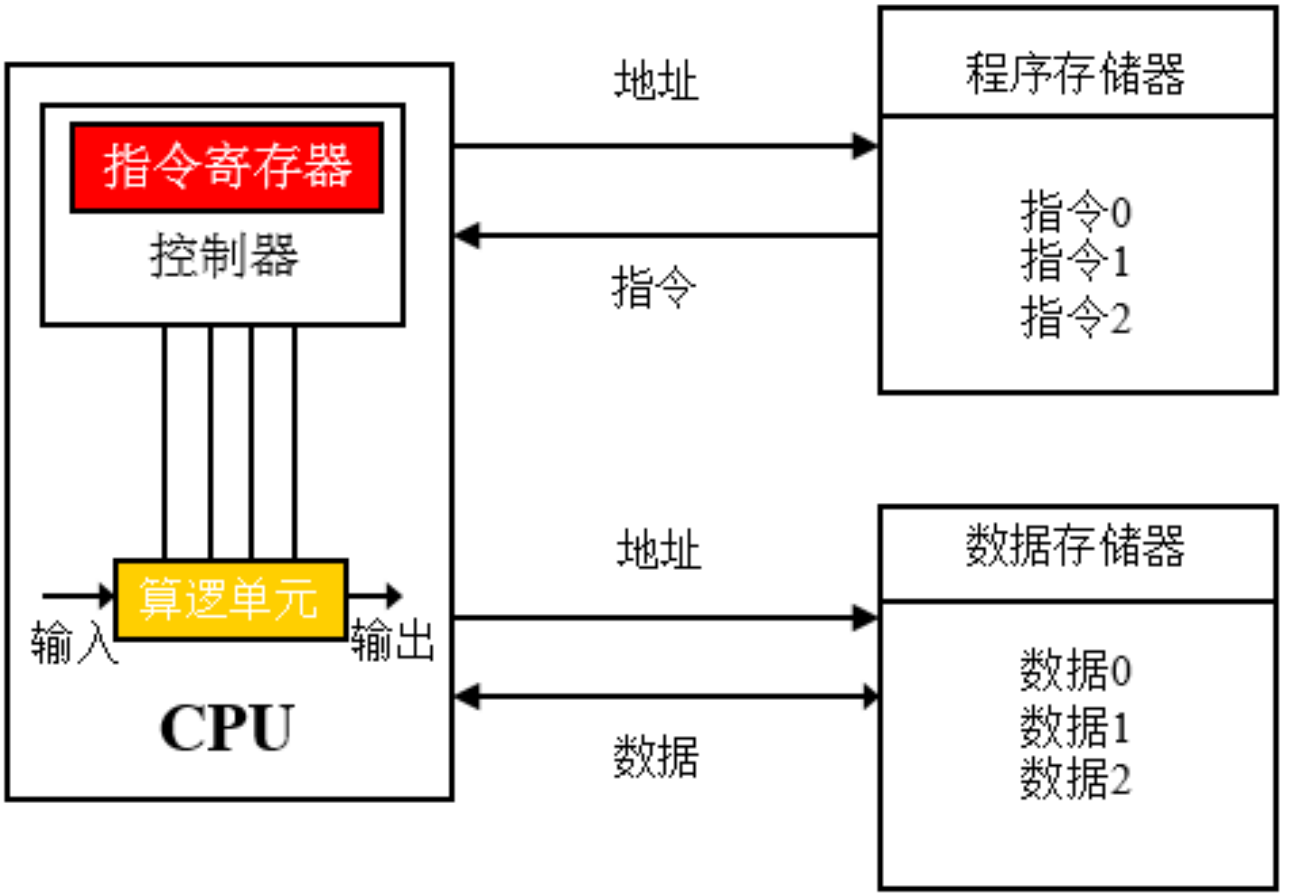
冯 诺依曼体系结构



冯 诺依曼体系结构


存储器
程序
指令0
指令1
指令2
指令3
指令4
数据
数据0
数据1
数据2
指令寄存器
控制器
算逻单元
输入
输出
中央处理器

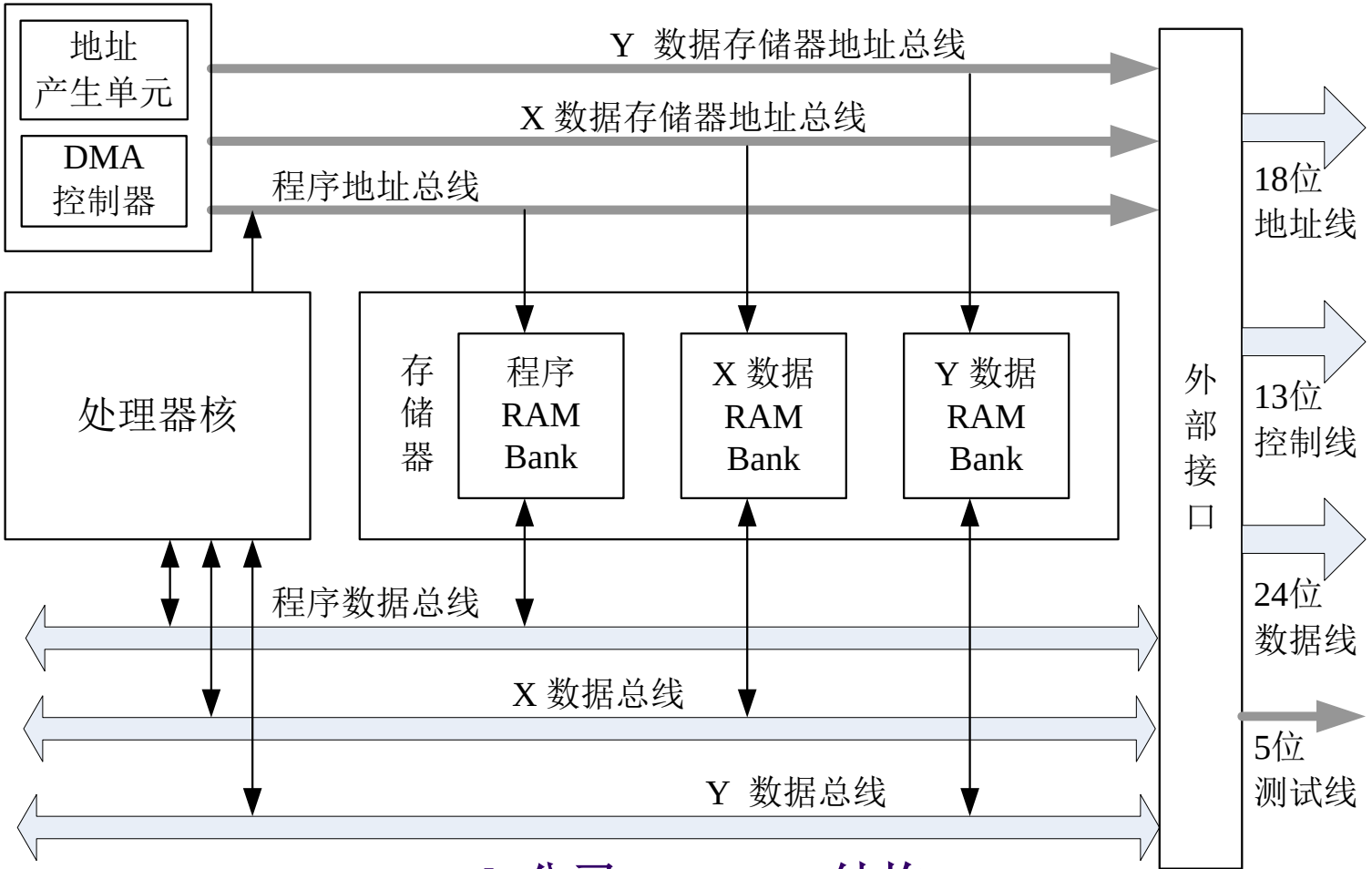
哈佛体系结构


指令寄存器
控制器
算逻单元
输入
输出
CPU
地址
指令
地址
数据
程序存储器
指令0
指令1
指令2
数据存储器
数据0
数据1
数据2

哈佛体系结构

- 哈佛结构基本特点：
  - ✓ 程序指令存储和数据存储分开的存储器结构。
  - ✓ 可以使指令和数据有不同的数据宽度。
    - 如Microchip公司的PIC16芯片的程序指令是14位宽度，而数据是8位宽度。
  - ✓ 优点：
    - 较高执行效率和数据吞吐率

哈佛体系结构



Motorola公司DSP56311结构

两种Cache结构

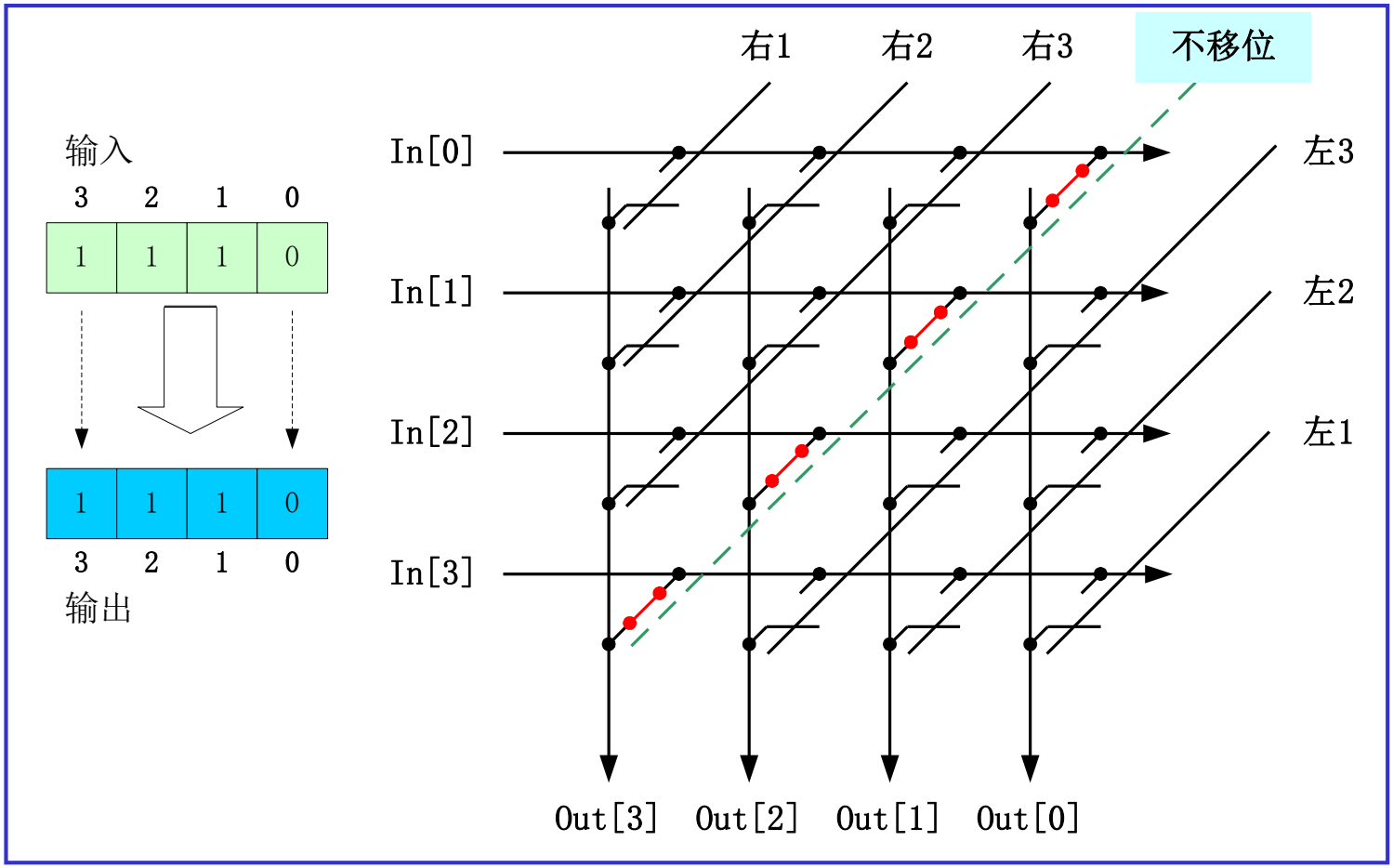
- 一种是数据和指令都放在同一个Cache中，称为普林斯顿结构或者统一化结构Cache（统一型Cache）。
- 另外一种是数据和指令分别放在两个独立的Cache中，称为哈佛结构（分离型）Cache
  - ✓ 优点：能够同时取指和取数；独立选择和优化cache的大小和结构
  - ✓ 缺点：程序通过写指令来修改程序自身的代码，两个Cache需要同时刷新；调整指令和数据Cache的比列

桶型移位器

- 通常的移位器都是一个时钟脉冲左移或者右移1位
- 桶型移位器采用了开关矩阵电路，可以做到用1个时钟脉冲移位任意位，如下图

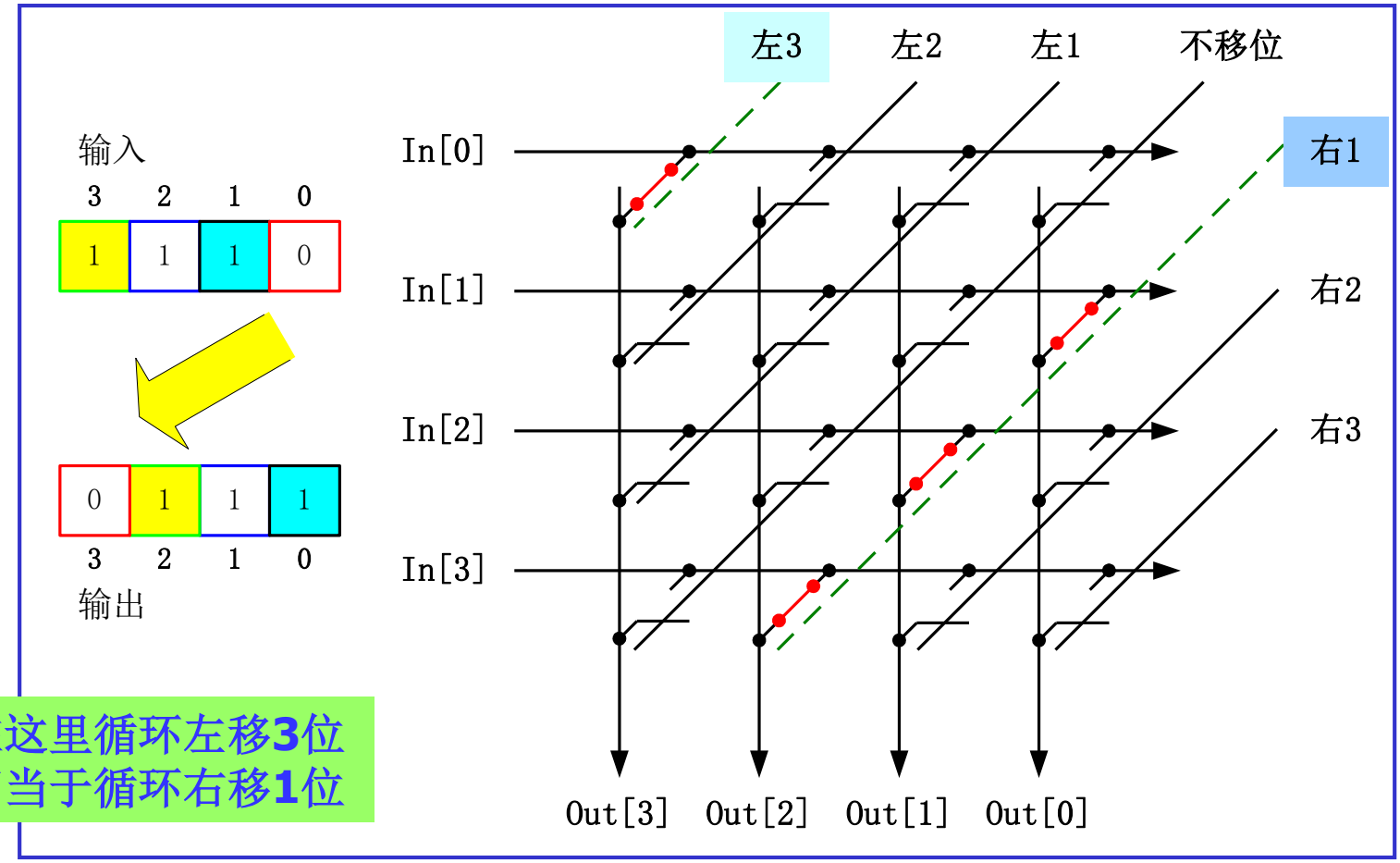
桶型移位器

不移位操作示意图



桶型移位器

- 循环左移3位操作示意图



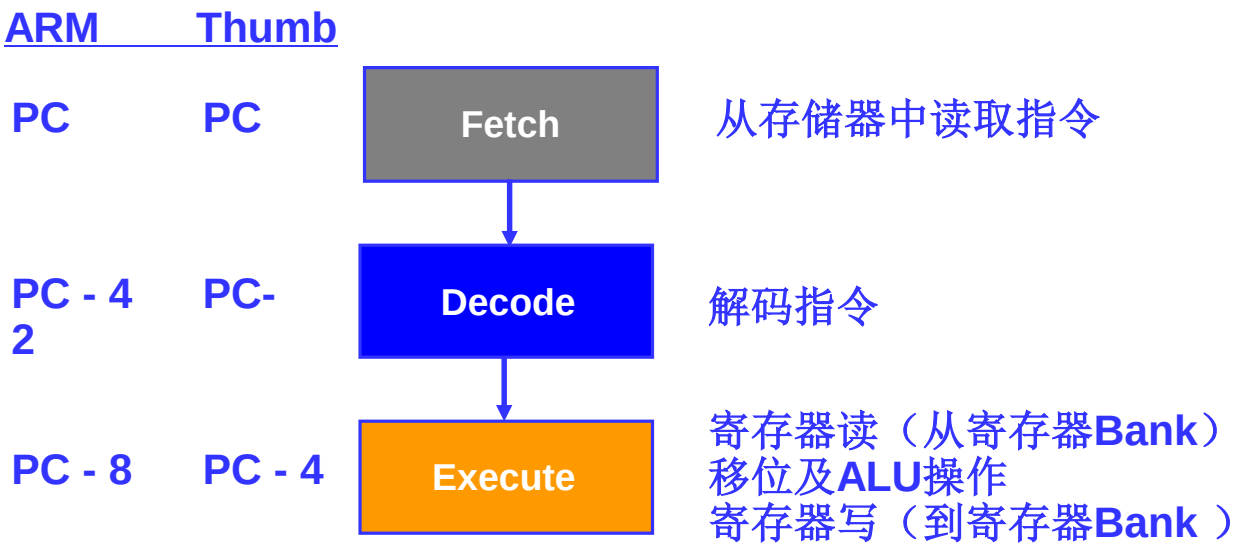
在这里循环左移3位
相当于循环右移1位

流水线技术

- 流水线(Pipeline)技术：几个指令可以并行执行
  - 提高了CPU的运行效率
  - 内部信息流要求通畅流动

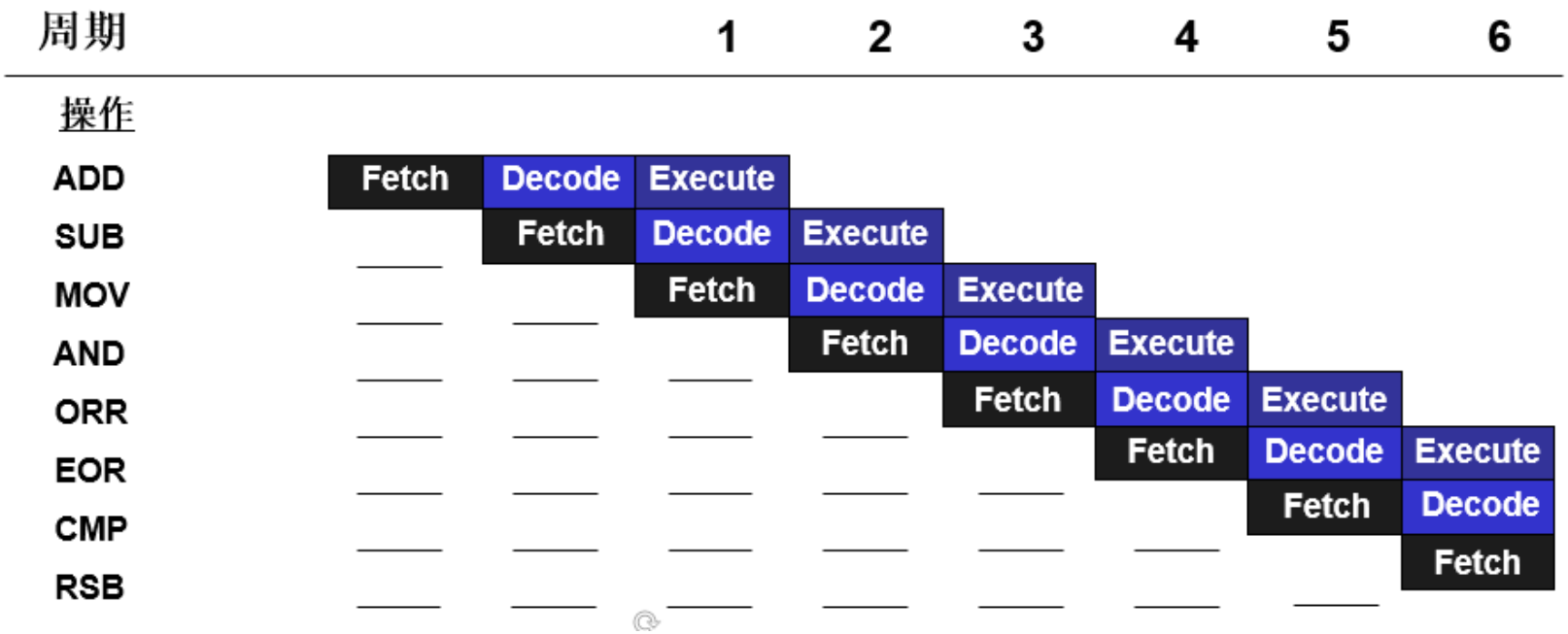
指令流水线—以ARM为例

- 为增加处理器指令流的速度，ARM7系列使用3级流水线.
  - ✓允许多个操作同时处理，比逐条指令执行要快。



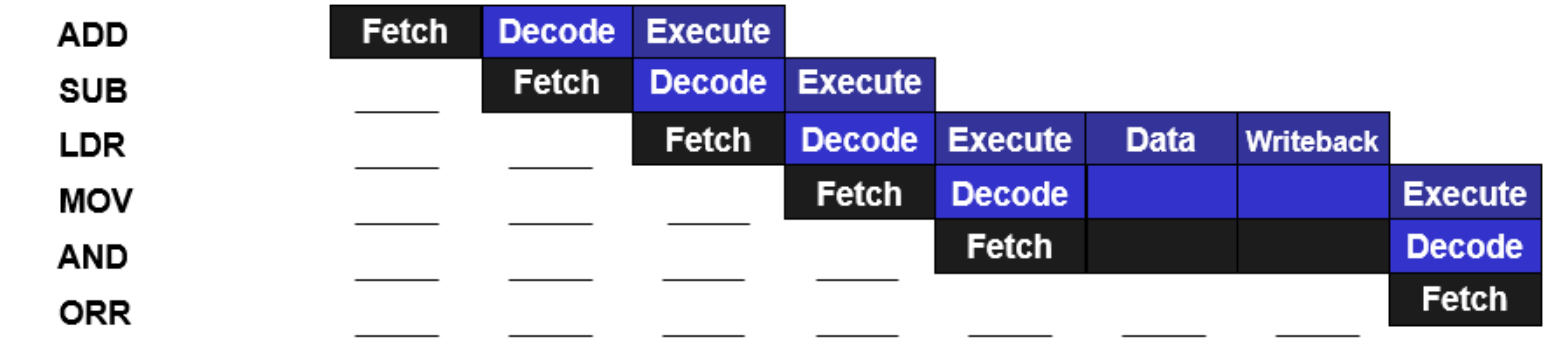
- PC指向正被取指的指令，而非正在执行的指令

最佳流水线



- 该例中用6个时钟周期执行了6条指令
- 所有的操作都在寄存器中（单周期执行）
- 指令周期数（CPI）= 1

LDR 流水线举例



- **LDR R2, [R3, #0x0C]!**
- 该例中，用6周期执行了4条指令
- 指令周期数（CPI）= 1.5

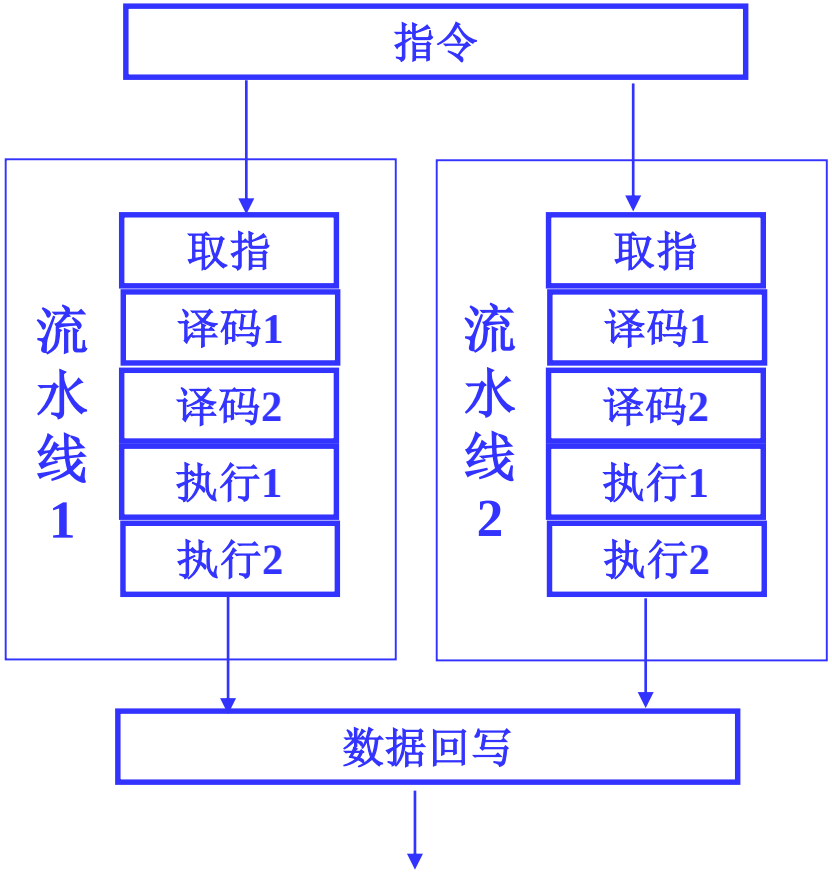
分支流水线举例

周期		1	2	3	4	5
地址	操作					
0x8000	BL	Fetch	Decode	Execute	Linkret	Adjust
0x8004	X		Fetch	Decode		
0x8008	XX			Fetch		
0x8FEC	ADD				Fetch	Decode	Execute
0x8FF0	SUB					Fetch	Decode	Execute
0x8FF4	MOV						Fetch	Decode
								Fetch

- 流水线被阻断
- 注意:内核运行在ARM状态

超标量执行

- 超标量(Superscalar)执行：超标量CPU采用多条流水线结构



谢谢！

Q & A