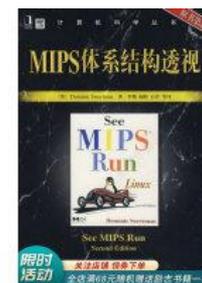
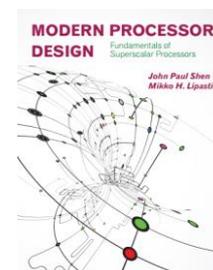
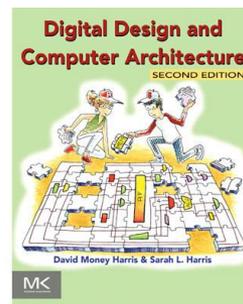


The RISC-V Processor Implementation: Pipeline——ILP

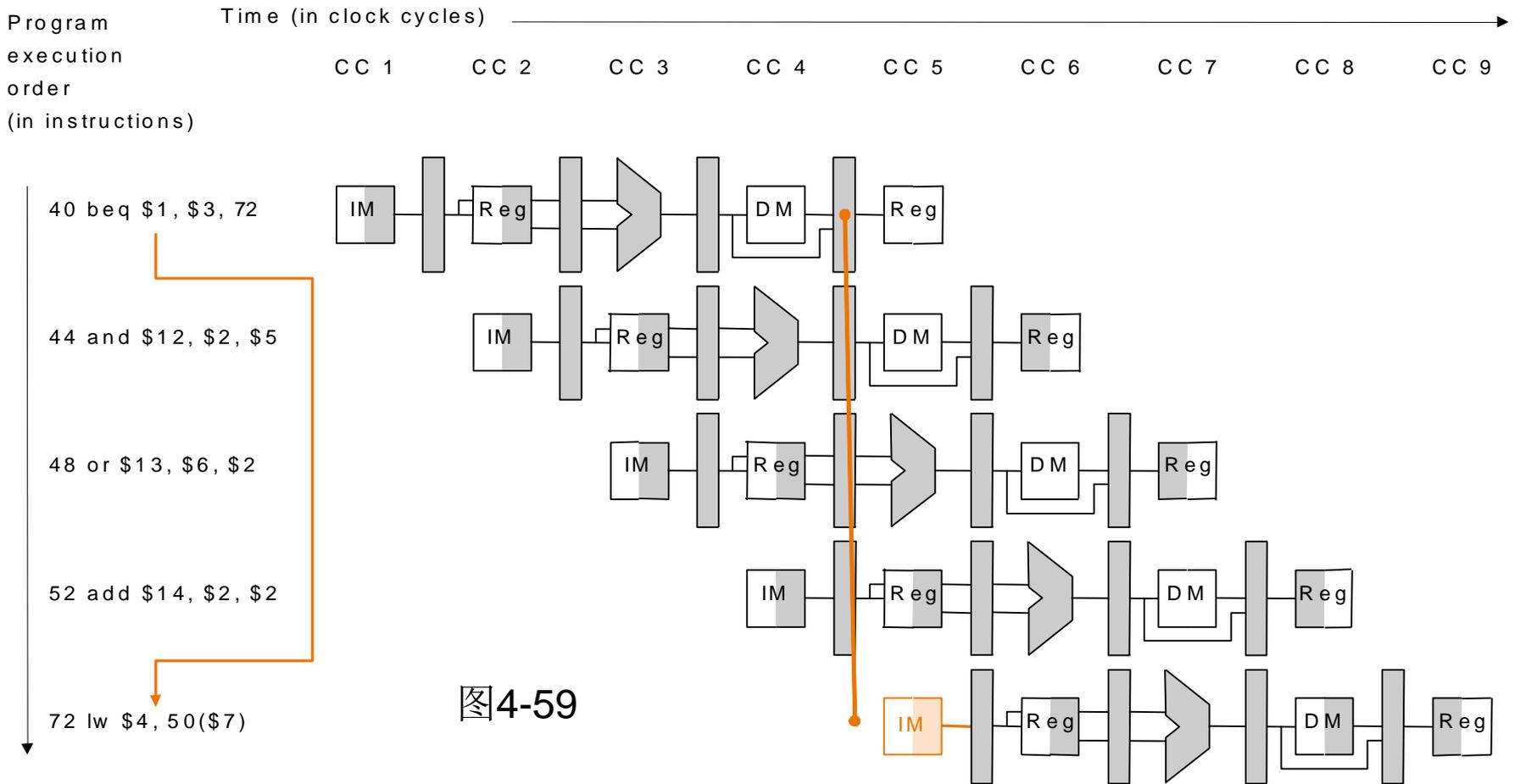
**“Computer Organization & Design ”
David Patterson, John Hennessy**

内容提要

- 流水线技术原理：4.6
- RV的五级流水线实现：4.6.1, 4.7
 - Verilog行为级定义：4.14
- Hazard问题：4.6.2
 - 结构冲突：哈佛结构
 - 数据依赖：4.8
 - 编译技术：插入nop, 指令重排, 寄存器重命名
 - forwarding技术：RAW
 - Interlock技术：Stall
 - 控制相关：4.9
 - 编译技术：延迟分支
 - 硬件优化：提前完成, 投机, 预测
 - 异常：4.10
- 多发射技术：4.11
- 硬件多线程：6.4
- 多核：6.5



beq taken?



beq指令完成时间？

- 以哪一条指令为基址？
- 能否在EXE完成？
- 是否有必要在EXE完成？

Step	R-Type	lw/sw	beq/bne	j
IF		IR = Mem[PC] PC = PC + 4		
ID		A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)		
EX	ALUOut = A op B	ALUOut = A + SE(IR[15-0])	If (A==B) then PC = ALUOut	PC = PC[31:28] (IR[25-0]<<2)
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR		

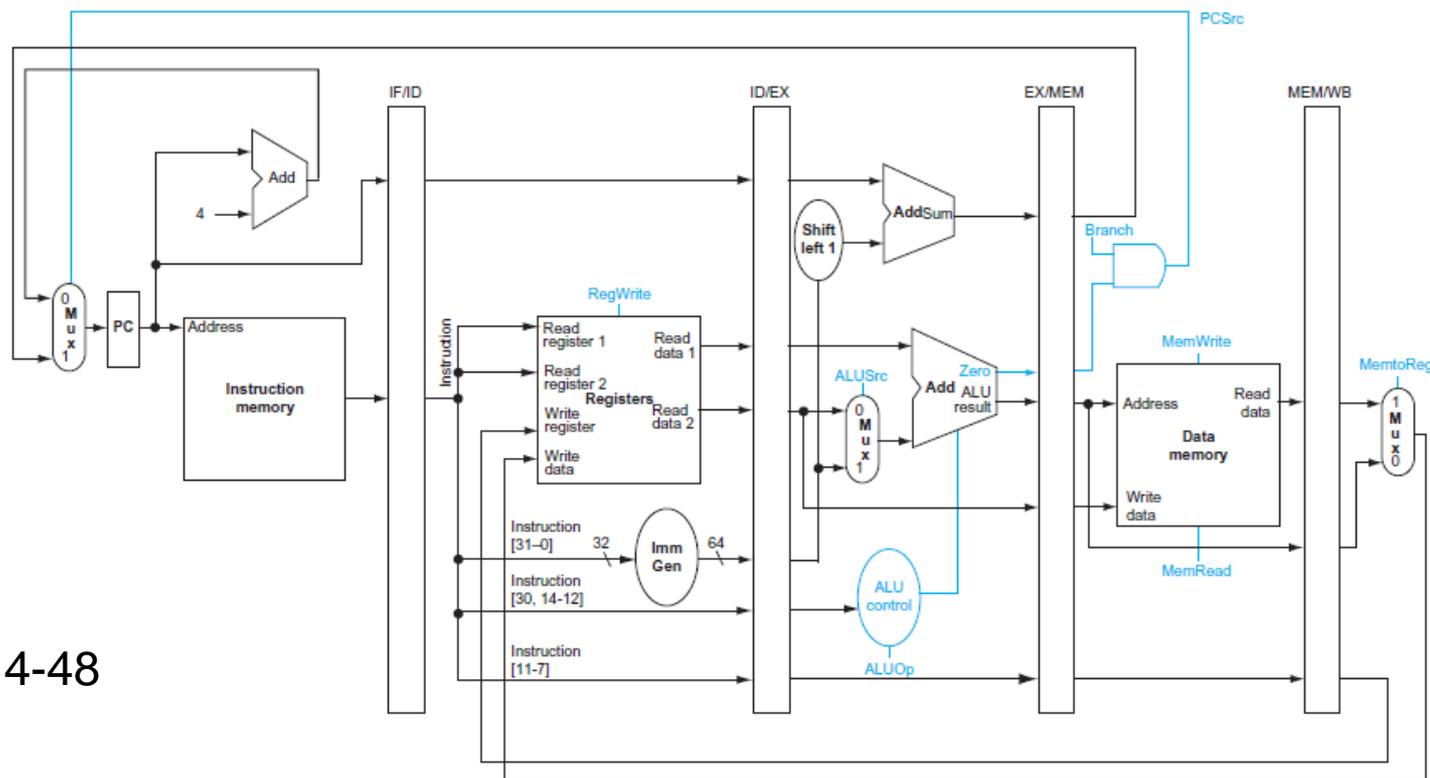
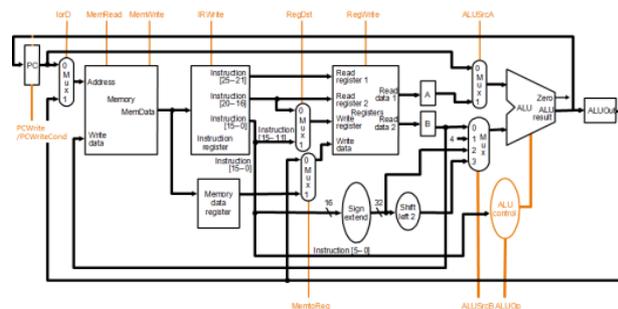
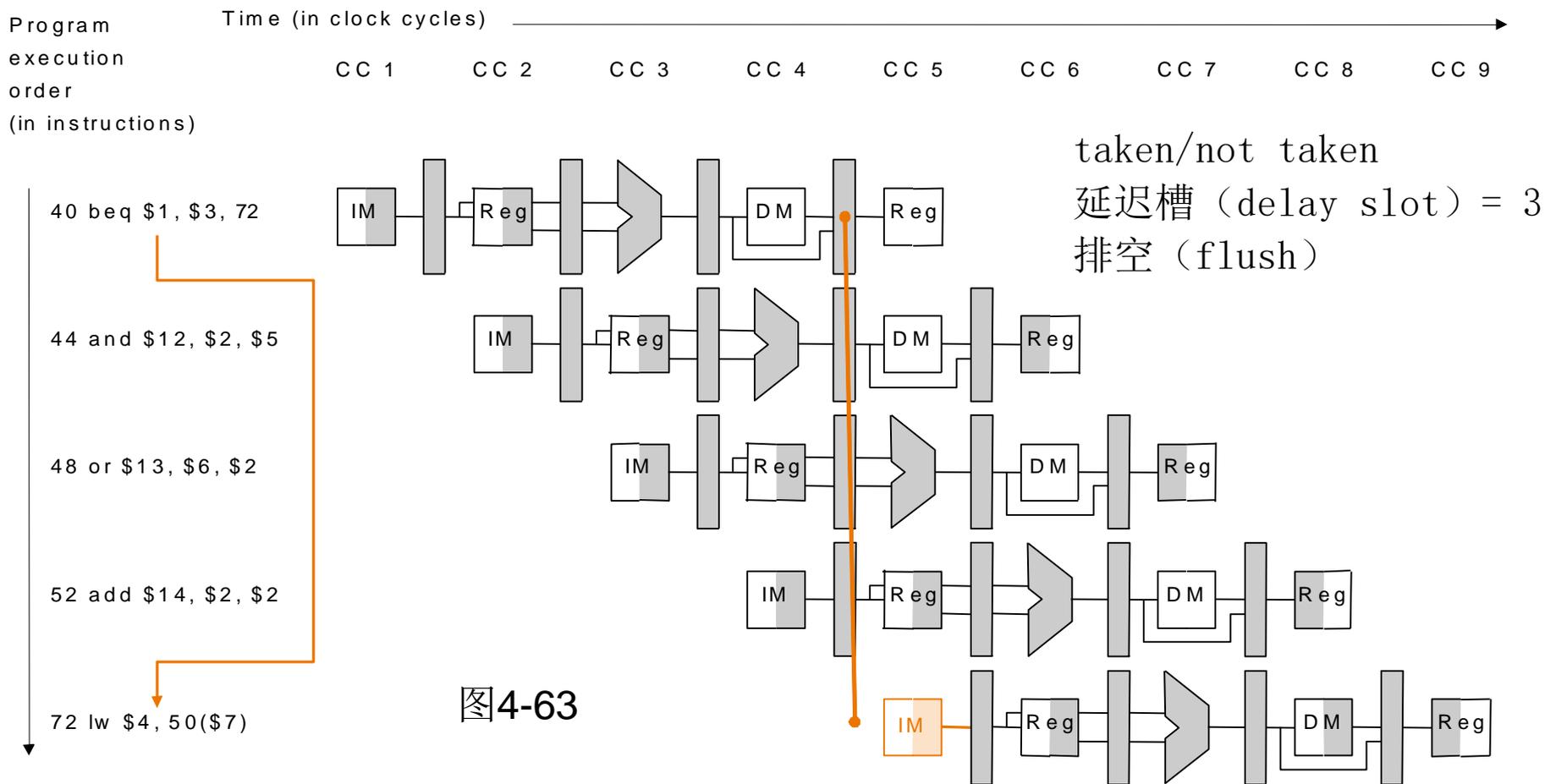


图4-48

beq branch hazard: stall?

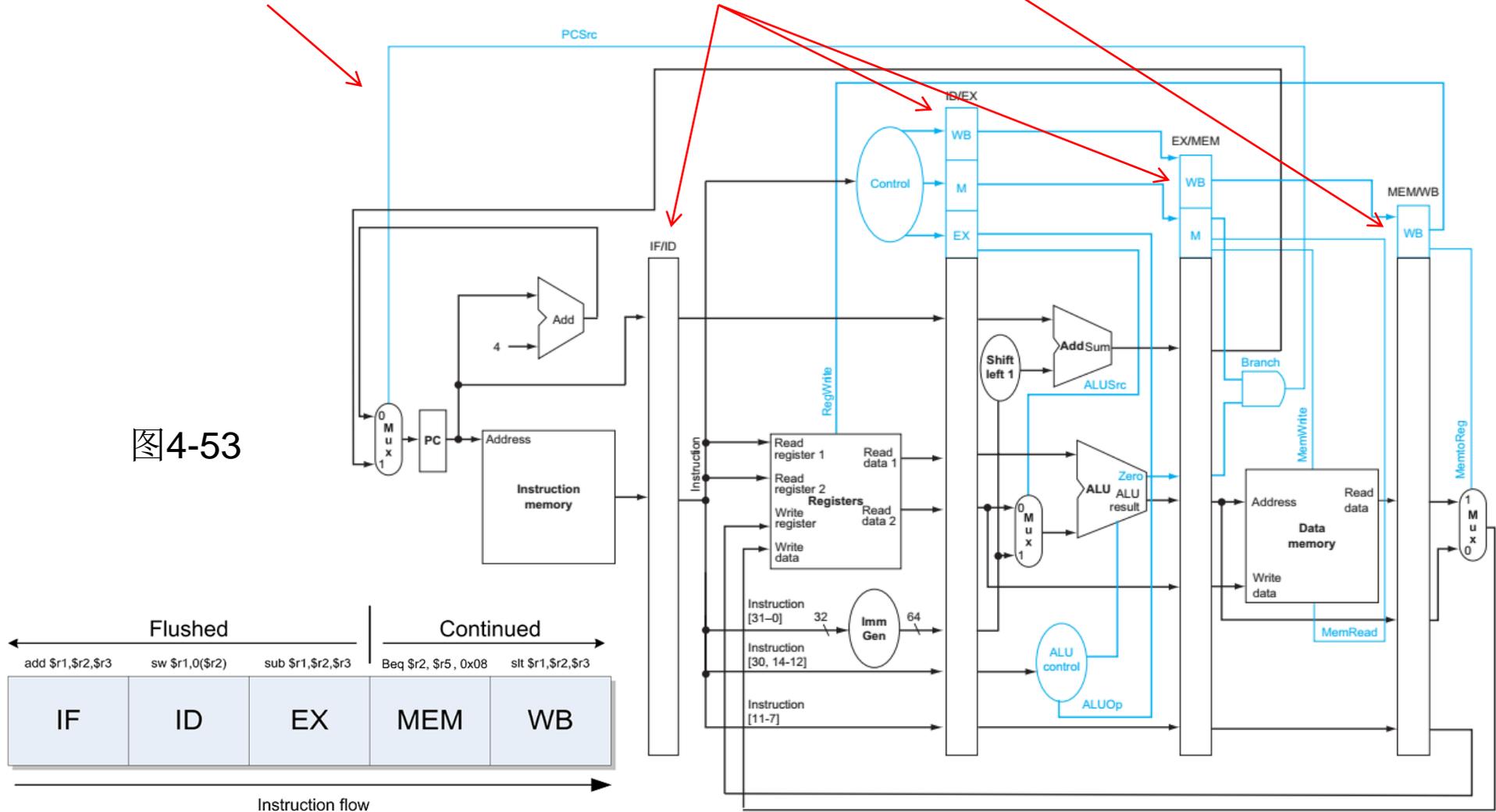


“相对于数据相关，分支频率低”，“没有好方案”！

beq: not taken与taken

taken: flush = Clear UP bubble down!

图4-53



减小beq损失：单周期，ID段，\$4.6.2, 4.9.2

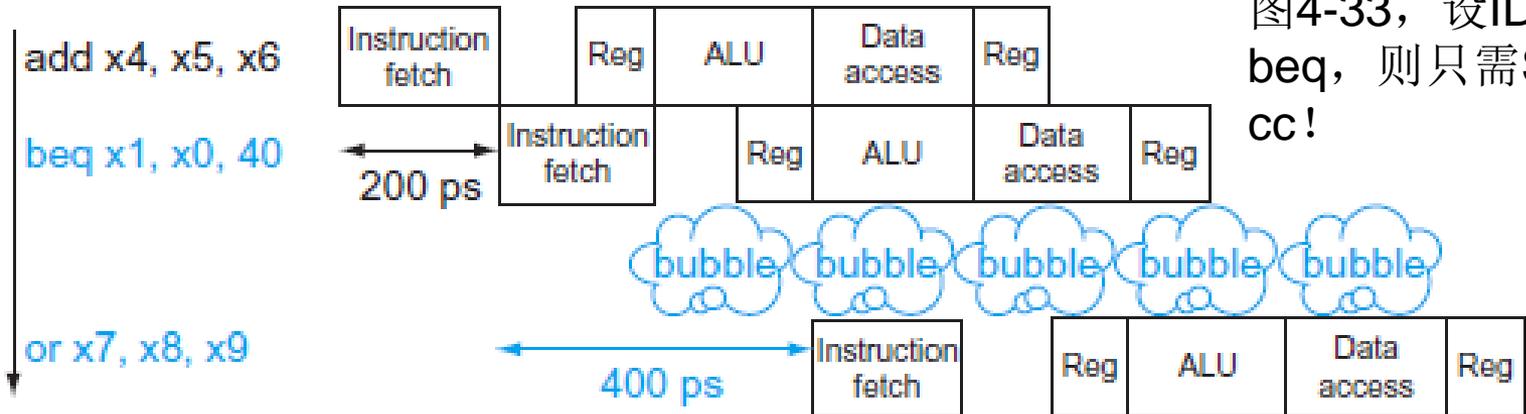
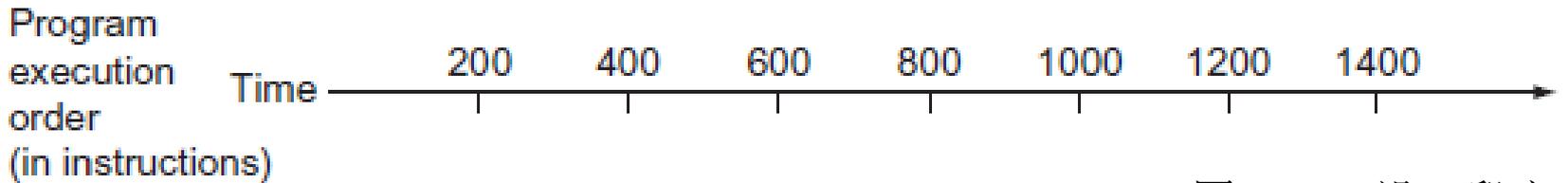
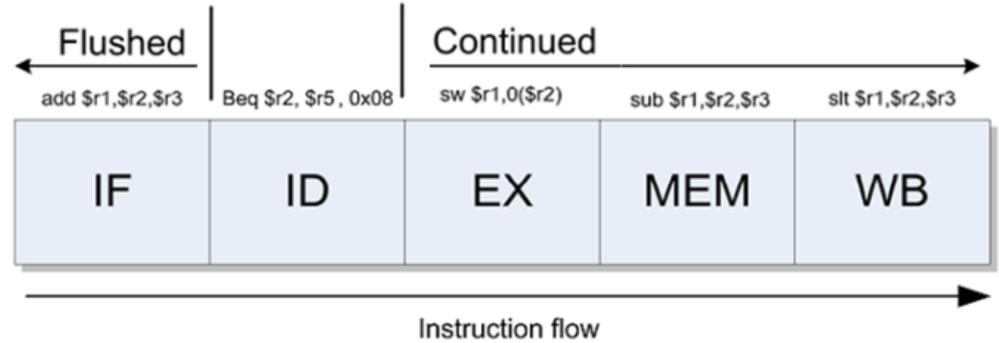
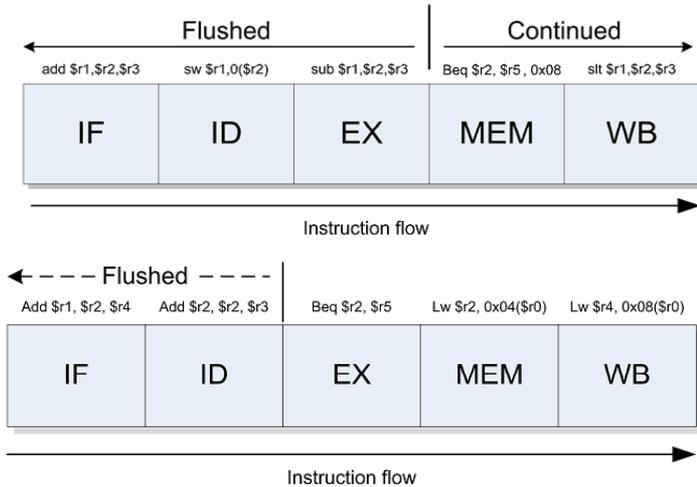
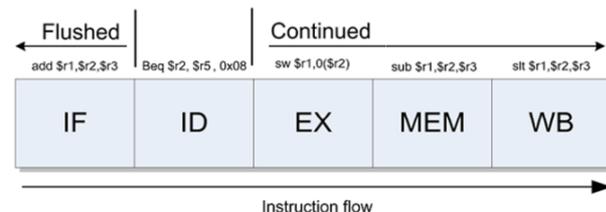
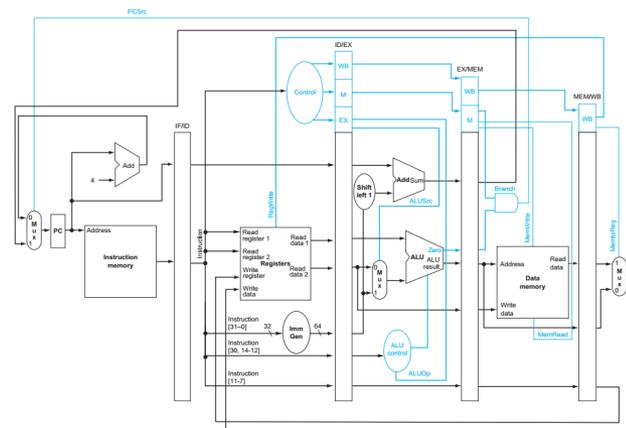


图4-33，设ID段完成beq，则只需Stall一个cc!

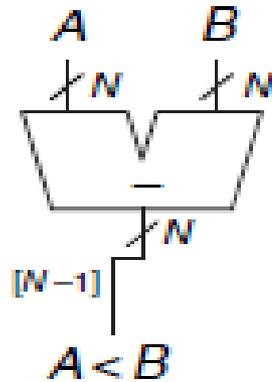
单周期分支实现

- 将beq完成时间从MEM提前到ID
 - 将加法器挪到ID段，计算nPC
 - 在ID段增加一个比较器，判断分支条件
 - 针对“简单条件判断”：相等比较器（按位xor，再or）
 - 不适于“复杂条件判断”：需要ALU计算
 - 假设分支**不发生**!!!
- 分支发生：只需更新PC和flush IF段
 - clear up: 增加IF.Flush，清空IF/ID（图4-66）
 - “由controler控制 IF.Flush”：应该与比较器联合控制吧？
 - bubble down: 向ID/EX送“0”
 - **损失**一个周期

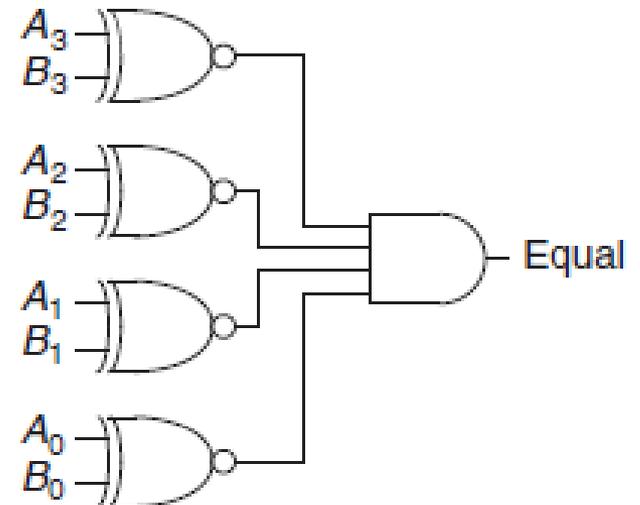
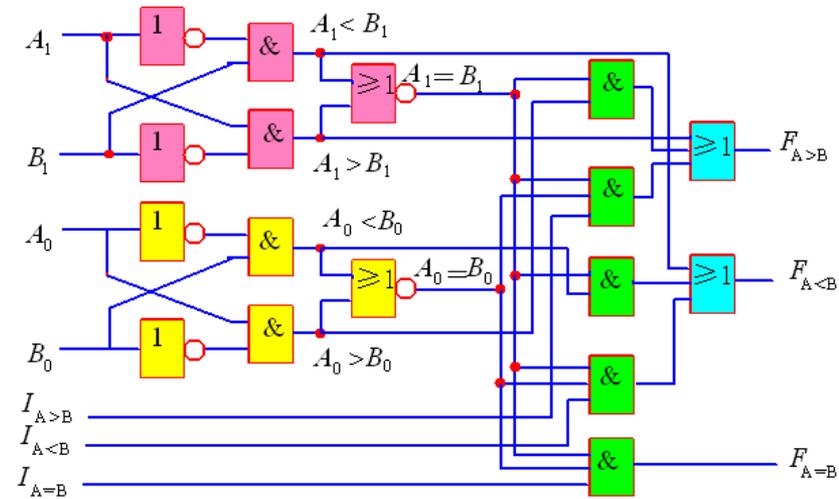
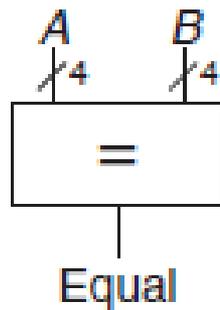


Comparator: 数值大小, 相等

- magnitude comparator



- equality comparator

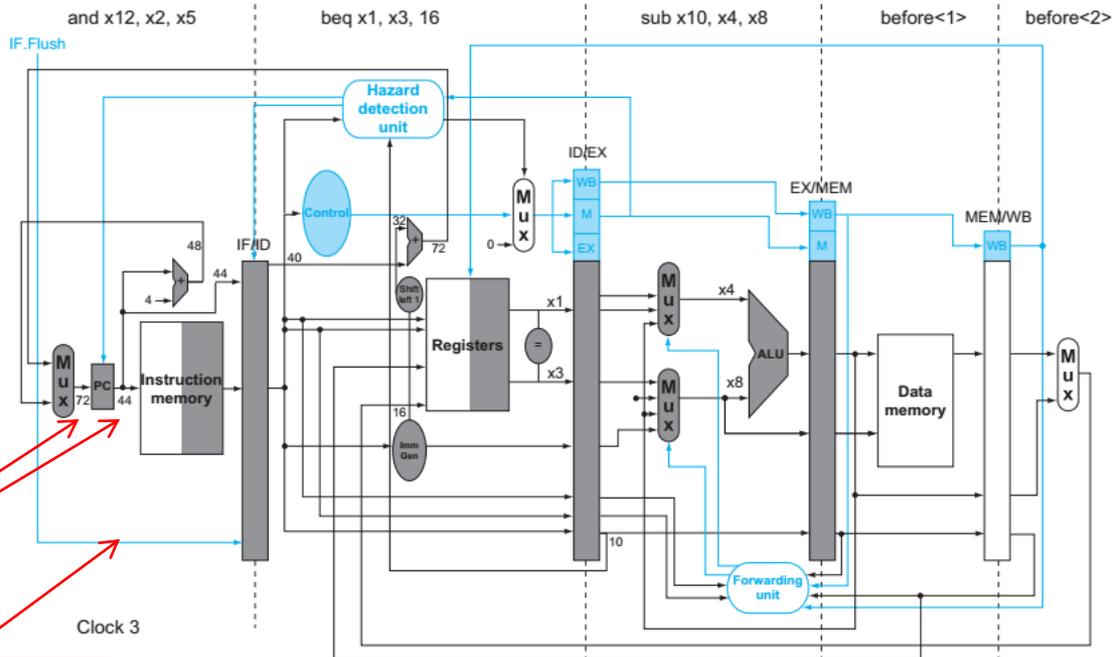


Flush & Taken

```

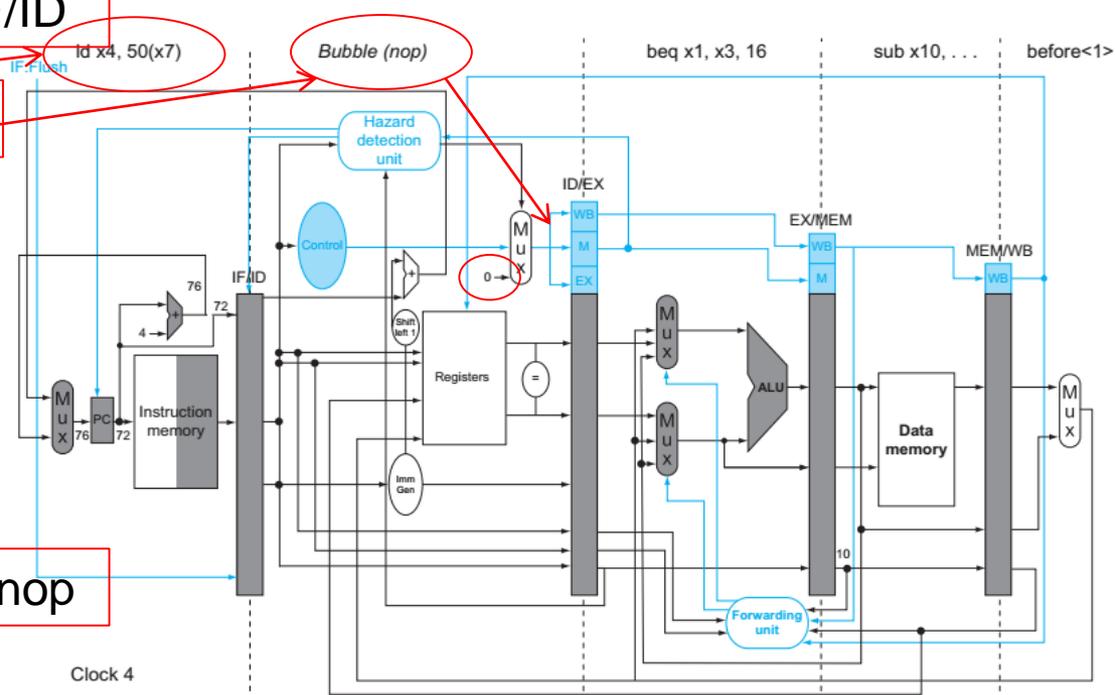
36 sub x10, x4, x8
40 beq x1, x3, 16
44 and x12, x2, x5
48 or x13, x2, x6
52 add x14, x4, x2
56 sub x15, x6, x7
...
72 ld x4, 50(x7)

```



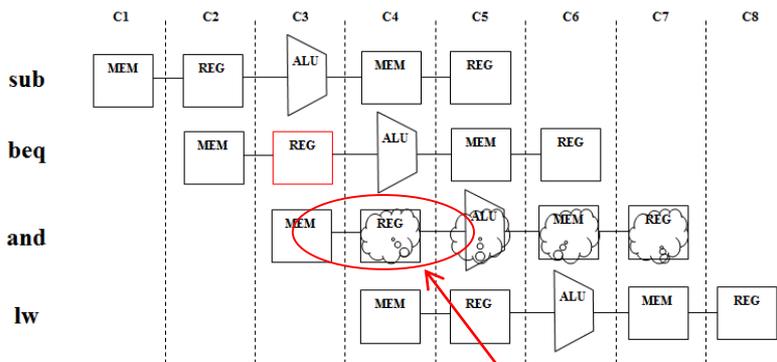
判定taken, 则: 更新PC, Clear IF/ID

取“ld”, bubble down



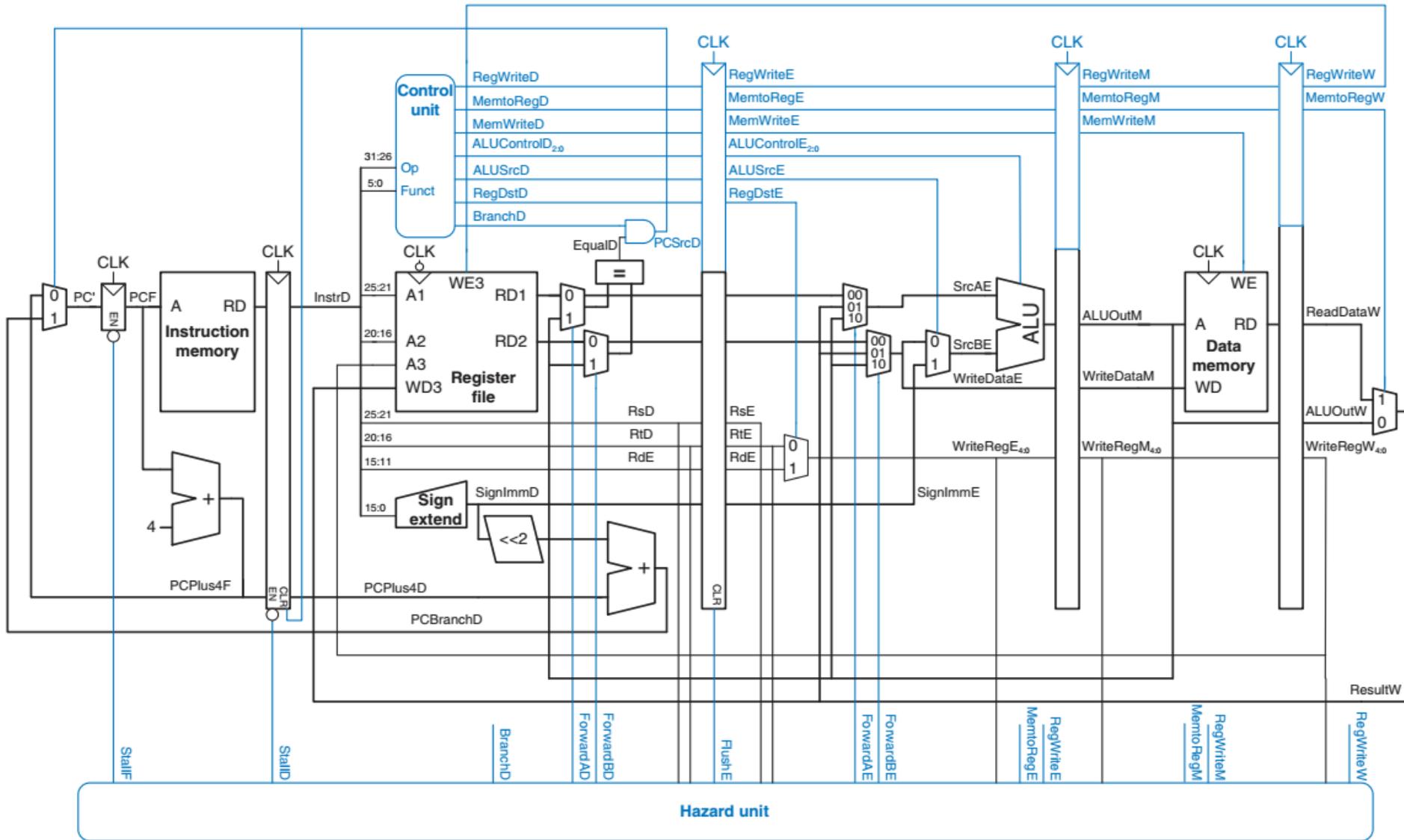
从cc4开始变为nop

• 图4-64



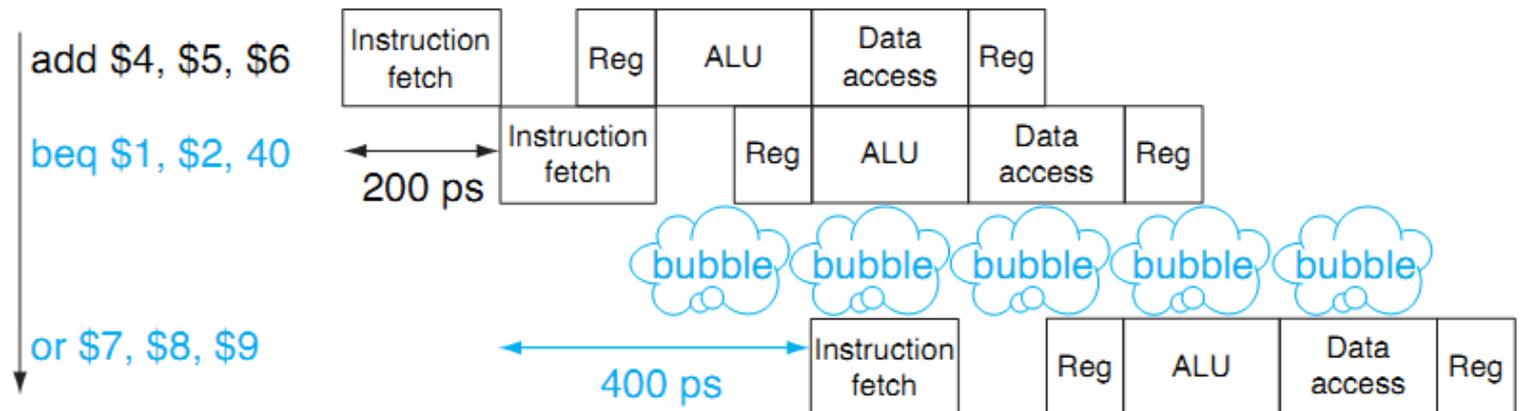
with full hazard handling

图 7.58



beq指令的性能， §4.6.1， p202

- 单周期分支stall对性能的影响， 例
 - SPECINT2006中， 条件分支占17%。 设其他指令CPI=1。
 - 单周期条件分支需stall一个周期， 因此平均CPI=1.17， 即性能下降1.17倍。
 - 原始版本（beq在MEM完成）的CPI=?



分支冒险的处理技术：硬件

- RV分支指令占比：36%（图2-48）
- 方法1 Stall：单周期beq减少stall，\$4.6.2，\$4.9.2
- 方法2 分支预测：发生？目标地址？\$4.9.3
 - 静态预测：假设taken/not taken，投机执行（Speculation）
 - 动态预测：使用程序运行时的实时信息预测，正确率>90%
 - 基于分支的局部和全局信息：是否发生，目标PC，发生方向
- 方法3 延迟分支（*delayed branch*）：软硬件，\$4.6.2
 - 编译器按一定的模式向延迟槽（*delay slot*）填入无关指令
 - 延迟槽总是被执行，不管分支发生与否，且先于分支指令提交
 - 向前找？向后找？——三种模式（见COD5-MIPS）
 - 一般单延迟槽（多个周期效率低，用动态预测）
 - MIPS采用单延迟槽（见COD5-MIPS），RV不支持！
- 方法4 条件执行：ARM

分支预测：静态预测，投机执行

- 三类

- 总是不会发生：beq

- 符合顺序执行语义：简单，经济，有效
 - 利于Cache

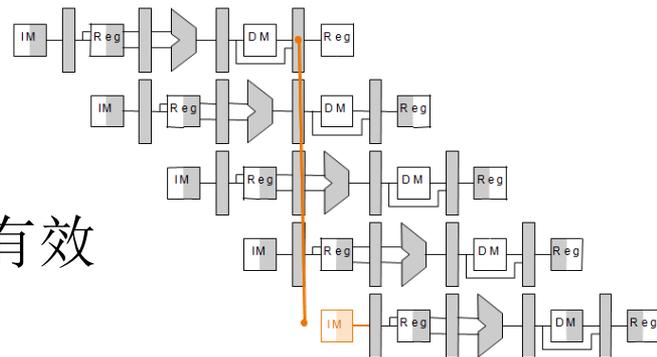
- 总是发生：bne

- 向后发生，向前不发生

- 与循环结构匹配：在循环中，向后转移用于迭代，向前转移用于退出循环。迭代次数多，退出只有一次。

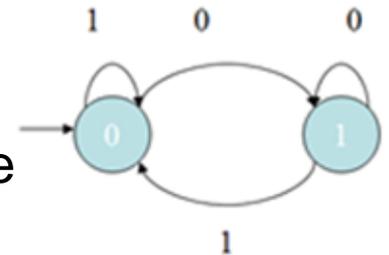
- COD的beq

- 假设not taken（准确率70%），且“单周期分支”

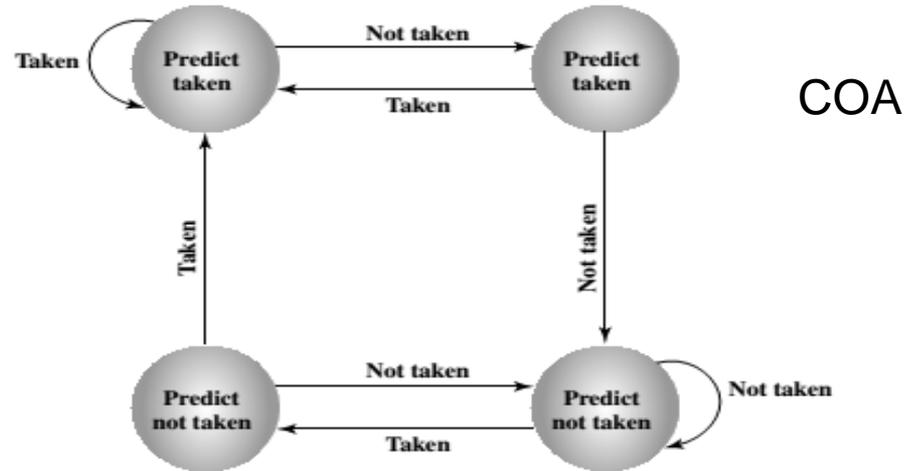
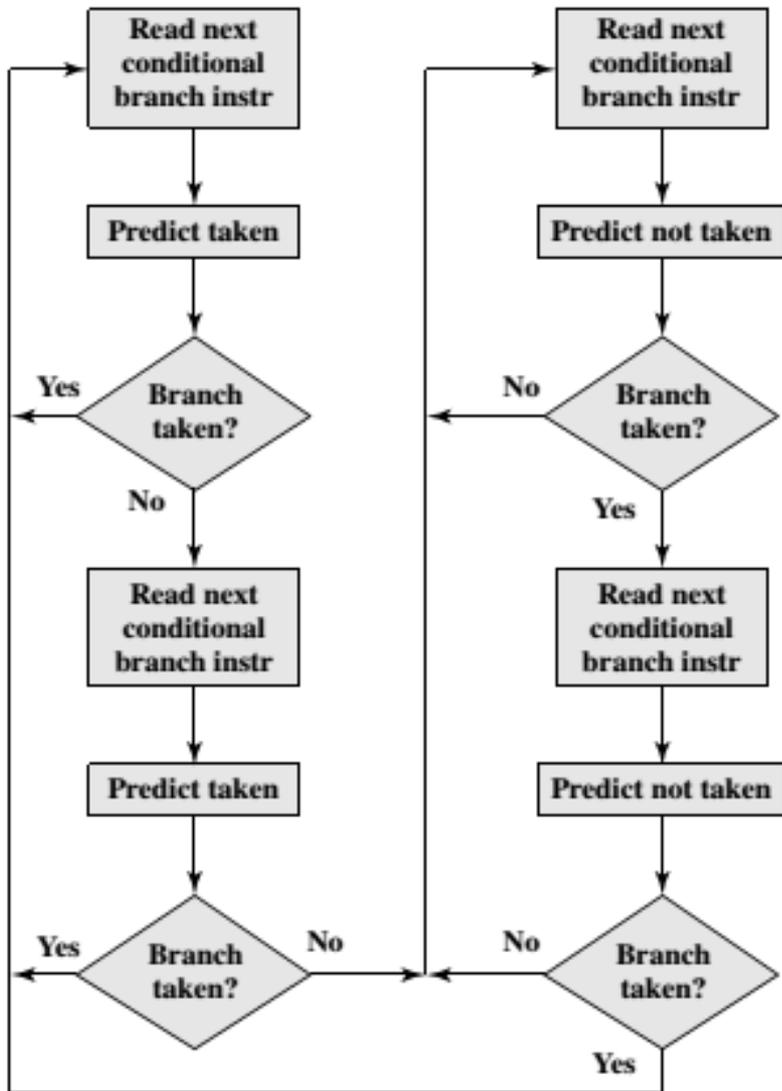


分支预测：动态预测，\$4.9.3

- 局部信息（BHT+BTB结合使用）：只考虑分支指令自己的历史
 - 分支预测器BHT：预测发生与否
 - 记录每条分支指令地址（低位，索引）和上一次分支状态（饱和计数）
 - 1位分支预测缓冲器，2位转移预测缓冲器
 - 取指时查找索引：命中，则按记录方向取指
 - 分支指令完成时按实际结果修正状态位
 - 分支目标缓存BTB：预测目标地址，直接映射Cache
 - 记录每一条分支指令的地址（地位）和上一次目标地址
 - 分支指令完成时按实际结果修正
 - 预测错误开销
 - in-order流水线：flush
 - out-of-order流水线：回滚（rollback）或恢复原始状态（undo）
- 全局信息：利用分支指令之间的关联性
 - 结合其他分支的记录。硬件开销大。
 - correlating predictors：每个分支使用两个2位预测器
 - Tournament Branch Predictor：每个分支使用多种预测器



Branch Prediction State Diagram



COA

2位预测器:

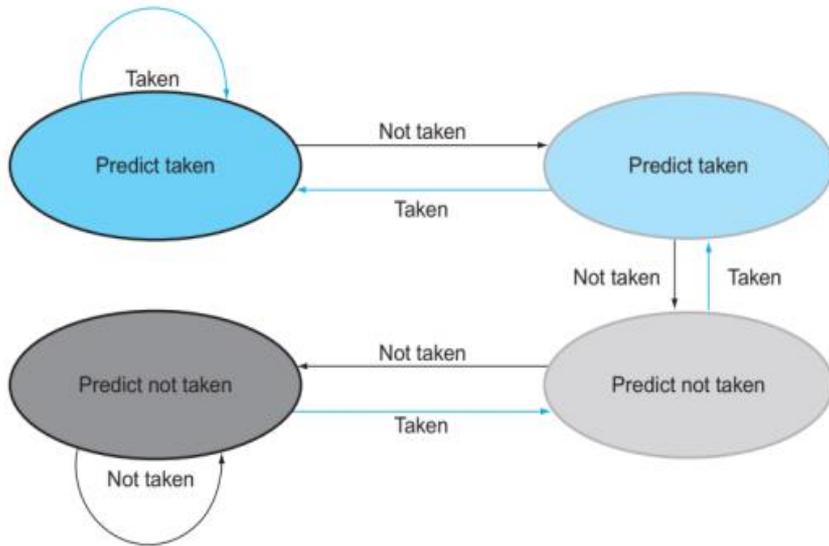
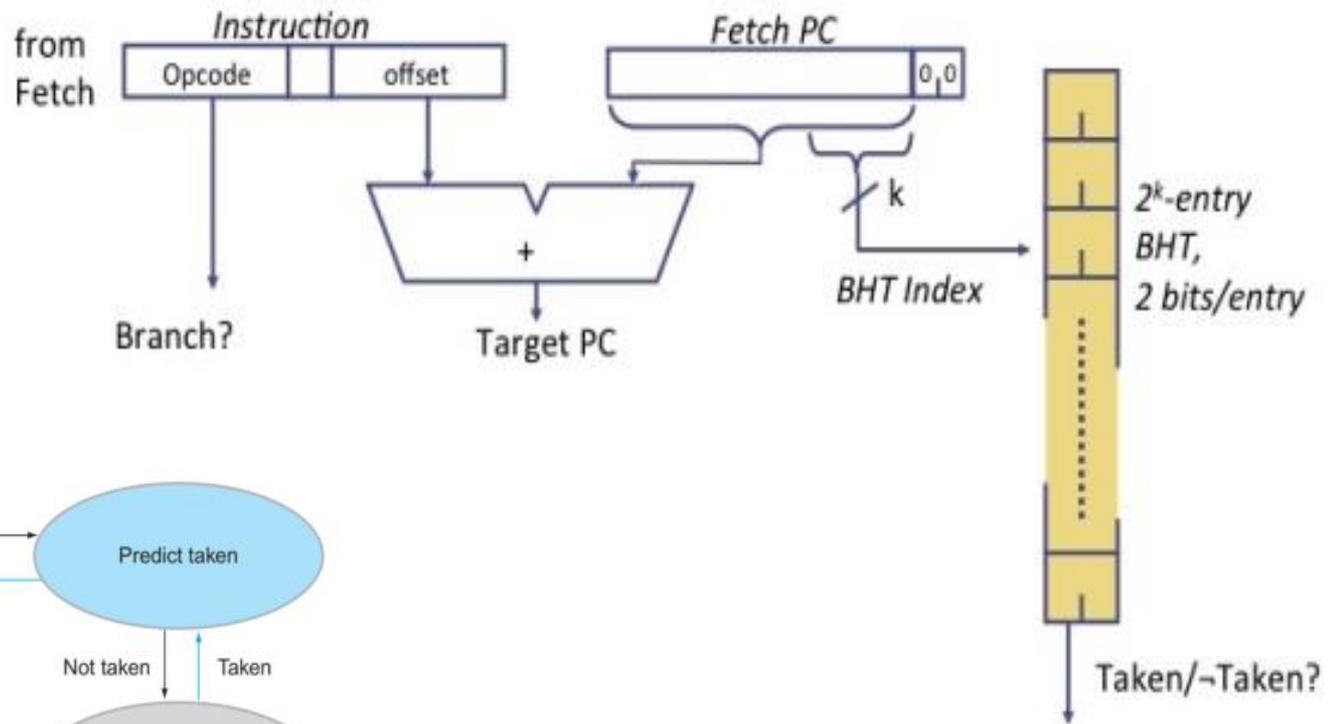
- 预测taken, 两次失败后预测not taken;
- 再预测not taken, 两次失败后转向预测taken。

Figure 12.18 Branch Prediction Flowchart

COA

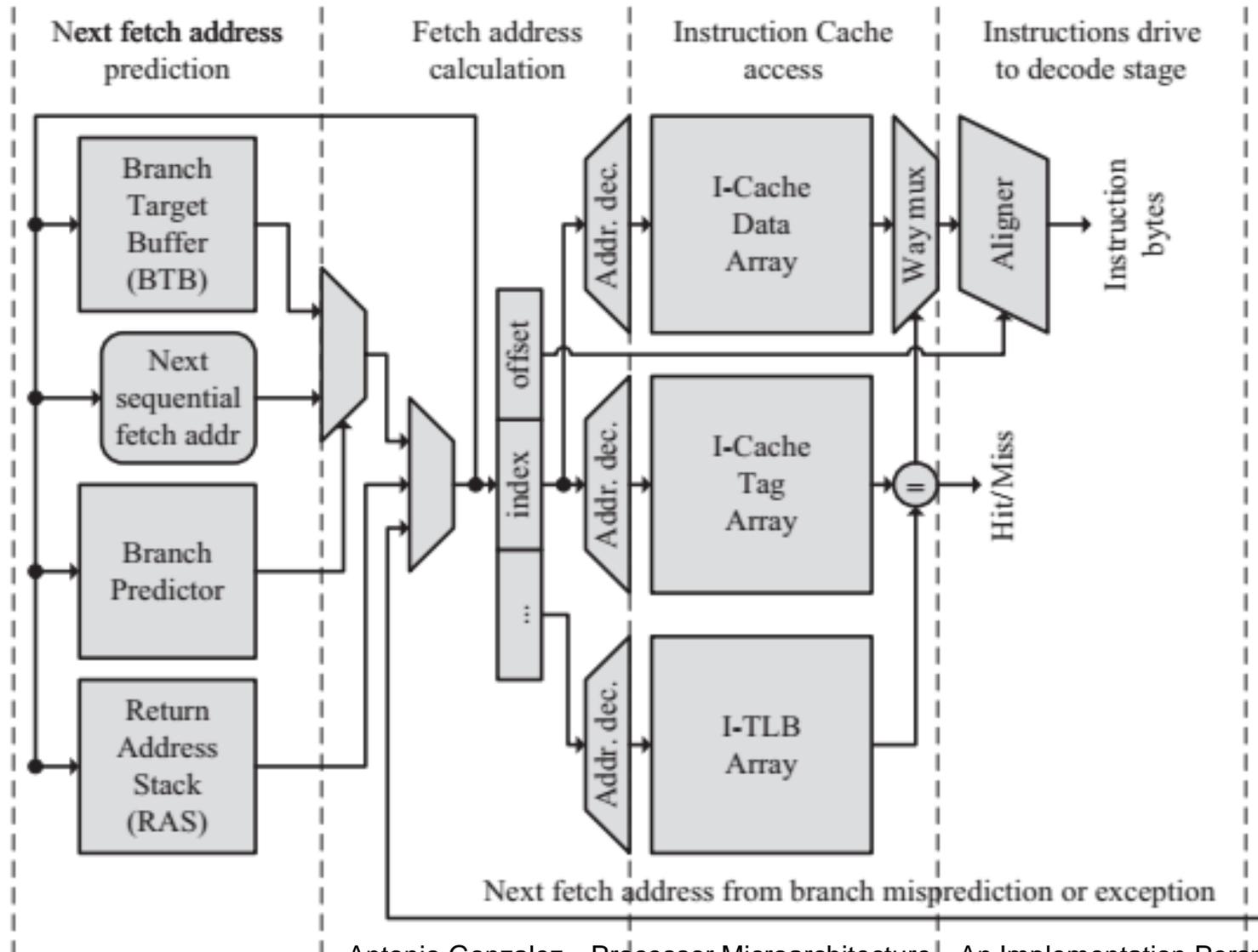
BHT (branch history table)

RV图4-65



- 以PC (即fetch PC) 的低 k 位为BHT索引
- Taken: PC+offset为NPC

Branch Prediction实现



发挥分支预测器的效益

```
if (data[c] >= 128)
    sum += data[c];
```

```
data[] = 226, 185, 125, 158, 198, 144, 217, 79, 202, 118, 14, 150, 177, 182, 133, ...
branch = T, T, N, T, T, T, T, N, T, N, N, T, T, T, N ...

= TTNTTTTNTNNTTTN ... (基本上随机出现 - 很难预测)
```

T = 该分支被选中
N = 该分支没有被选择

```
data[] = 0, 1, 2, 3, 4, ... 126, 127, 128, 129, 130, ... 250, 251, 252, ...
branch = N N N N N ... N N T T T ... T T T ...

= NNNNNNNNNNNN ... NNNNNNTTTTTTTTTT ... TTTTTTTTTT (很容易进行预测)
```

- 软硬协同：数据预排序，提升分支预测的有效性

动态调度流水线：OoO, §4.11.3

- 程序序 (program order)

- 控制流
- 数据流 (RAW)
- 按序执行 vs 乱序执行

```
ld x31, 0(x21)
add x1, x31, x2
sub x23, x23, x3
andi x5, x23, 20
```

- 动态分发dispatch: 减少数据依赖影响

- 结构
 - reservation station: 暂存op/opr
 - reorder buffer: 保存程序序, 暂存执行结果
- 行为
 - In-order issue: 进入RS和ROB
 - Out-of-order execute: 动态调度, 利用RS
 - WAR: 寄存器重命名
 - In-order commit: ROB保持程序序, 投机回滚, 异常处理

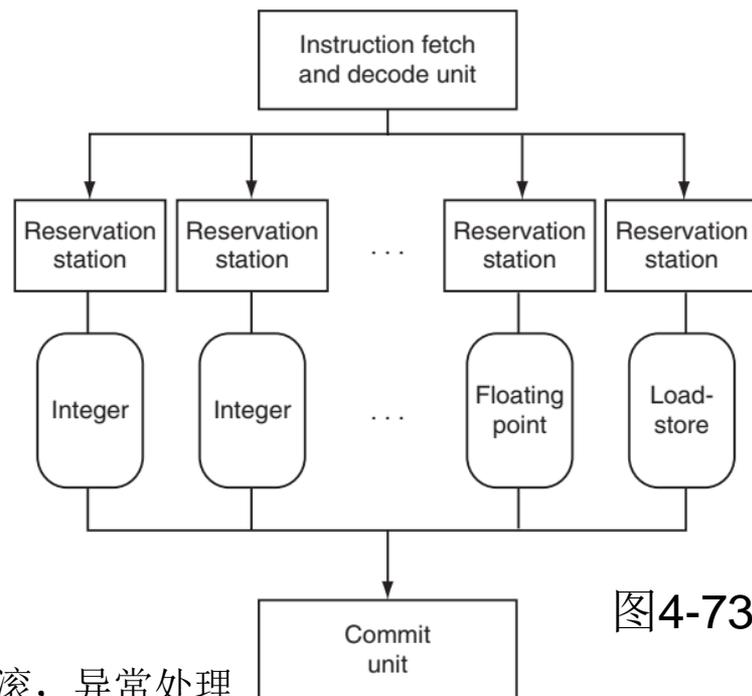
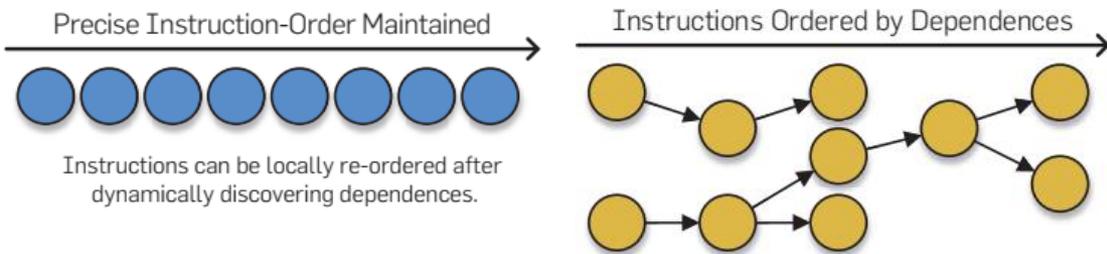
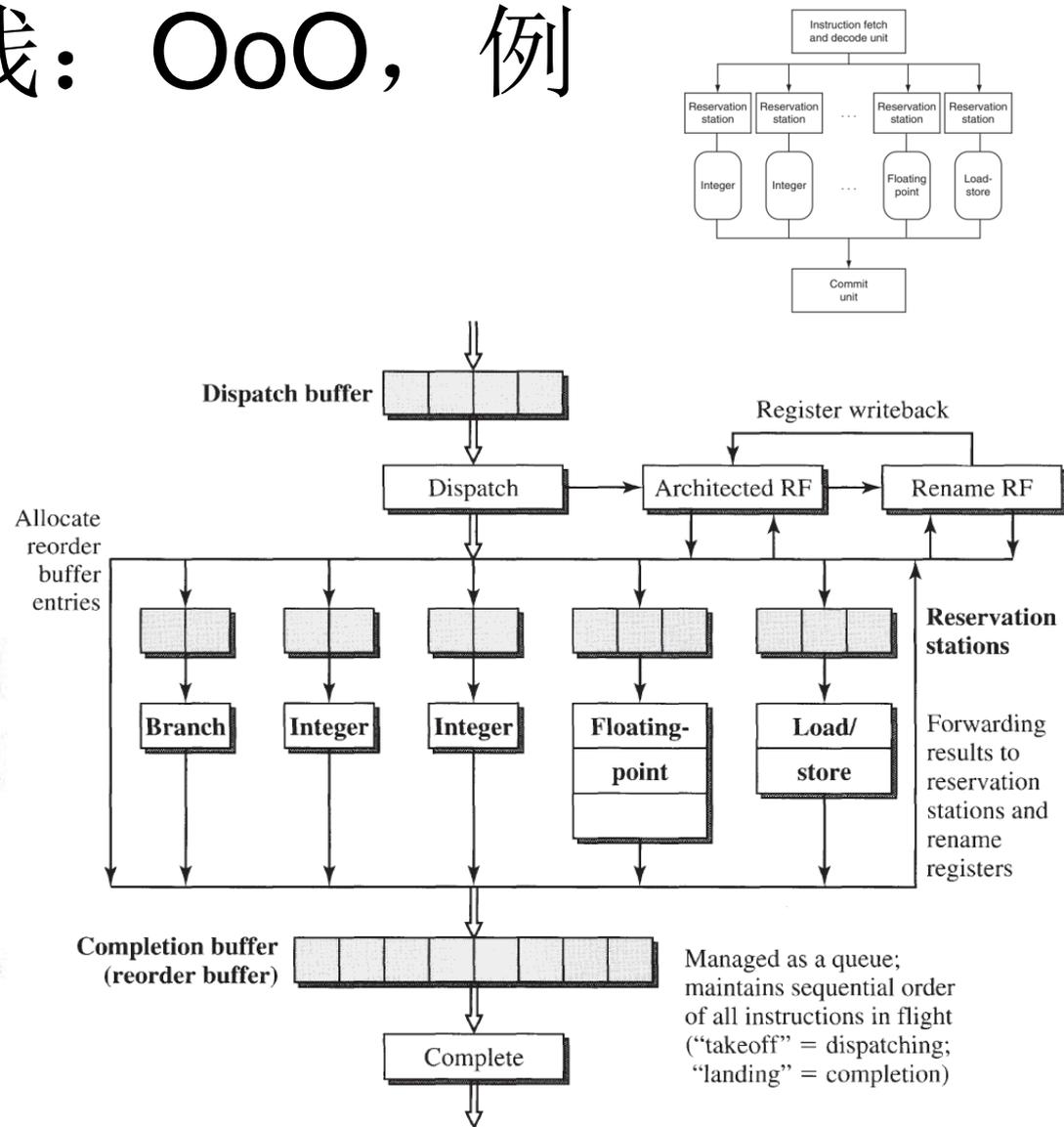
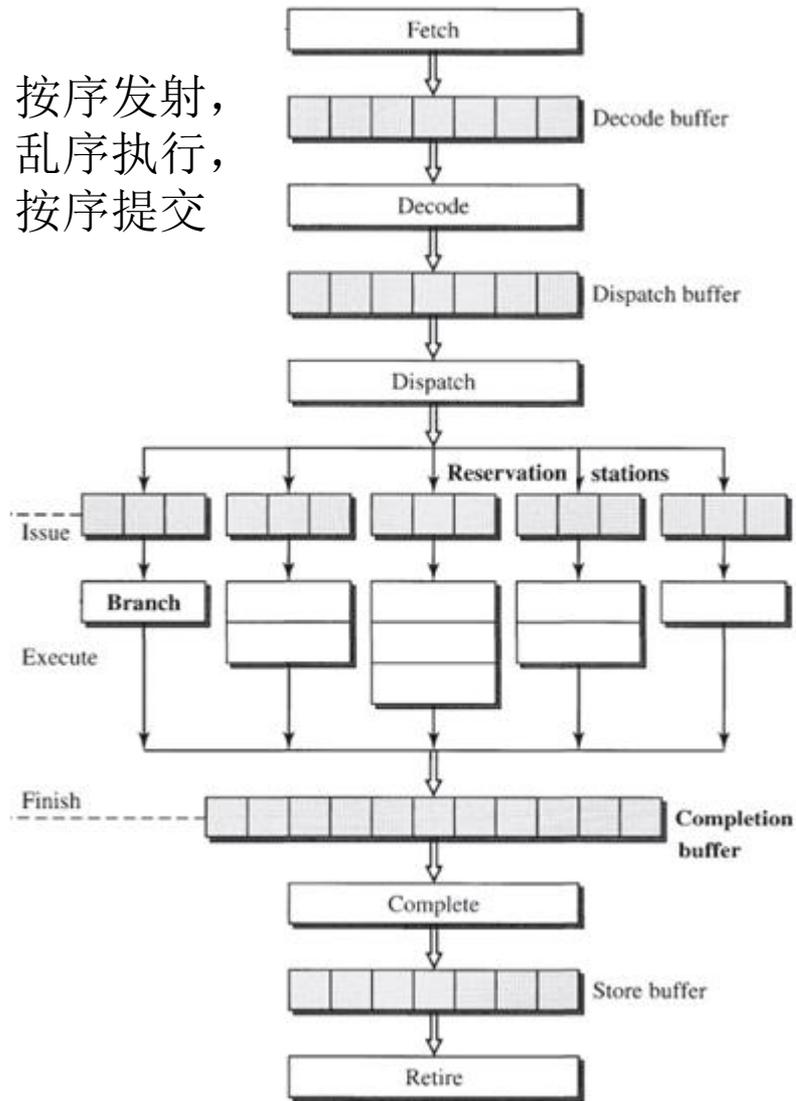


图4-73

动态调度流水线: OoO, 例

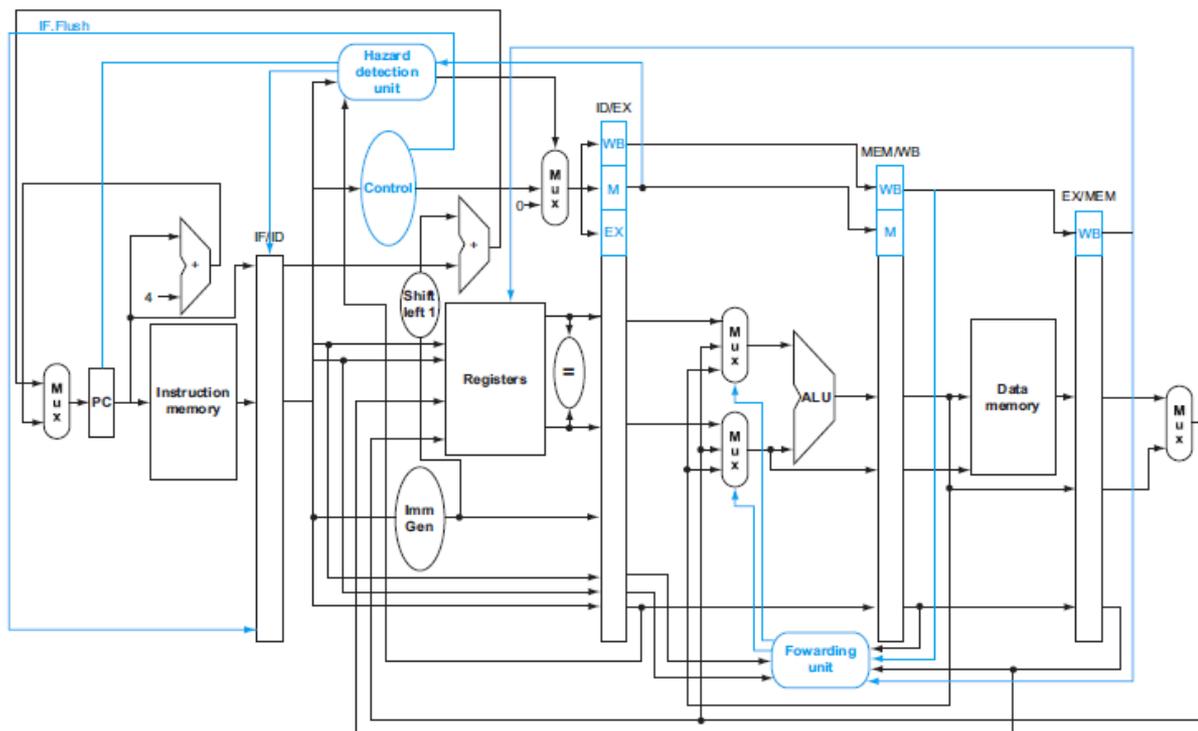
按序发射,
乱序执行,
按序提交



无条件分支？

- beq rs1, rs2, L1; PC相对, 12位offset
- jal x0, Label; J-type, PC相对, 20位offset
- jalr x0, 100(x5); I-type, 间接跳转

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]			rd	opcode	J-type
	imm[11:0]	rs1	funct3			rd	opcode	I-type



Benchmark: SPEC2006

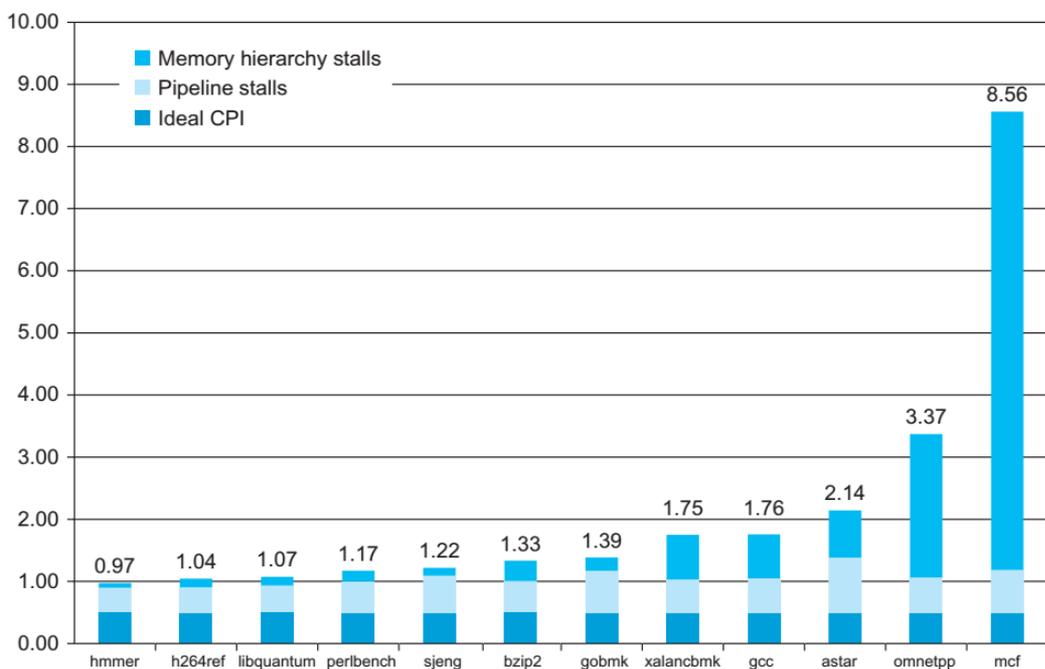


图4-73 CPI of ARM Cotex-A53

CPI: 理想0.5, 平均1.3

停顿: 60%冒险, 40%访存

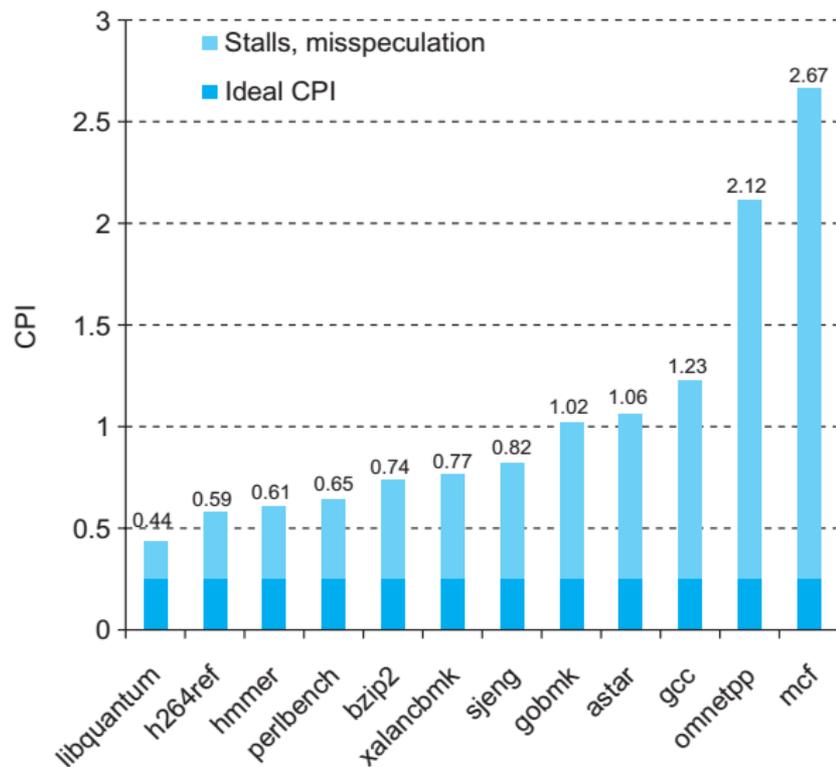
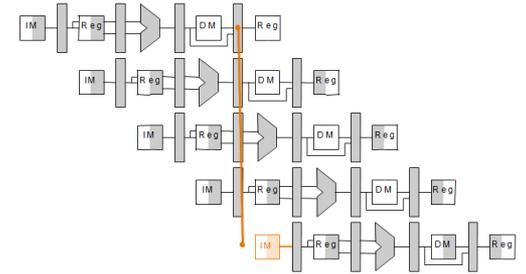


图4-75 CPI of Intel Core i7,

CPI: 理想0.25, 平均0.79

提高ILP, §4.11

- 提高k
- 降低CPI: $CPI \leq 1$
 - 推测/投机: 分支、st-ld依赖、值预测...
 - 恢复/回滚
 - 异常处理
 - 多发射: 增加“发射宽度”
 - 增加功能部件, 增加RF端口, 内存banked
 - 静态多发射: 编译器打包, VLIW (按序执行)
 - TMS320C64x
 - 动态多发射: 超标量superscalar
 - 动态调度流水线: 乱序执行



RISC多发射：数据通路的不对称性

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

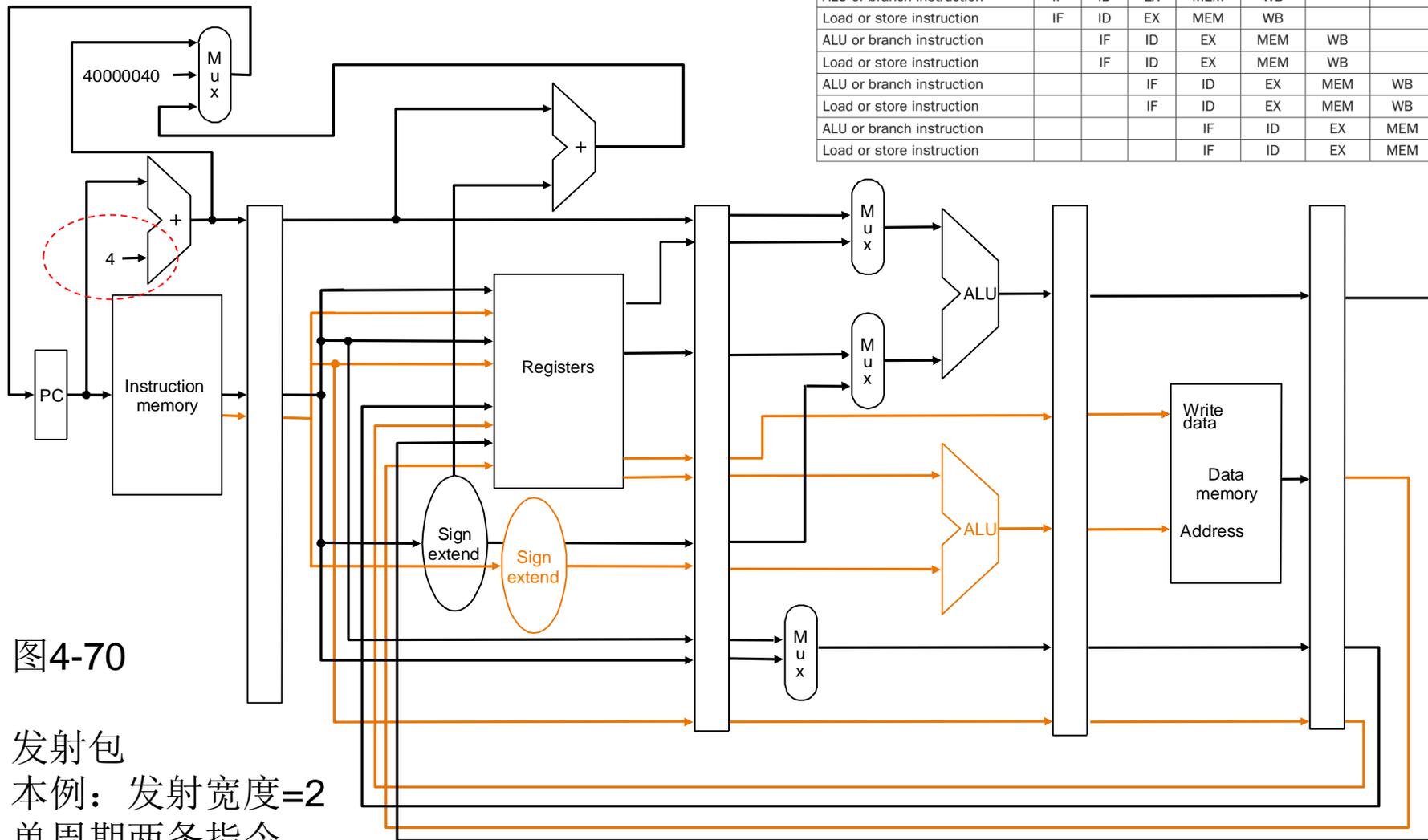


图4-70

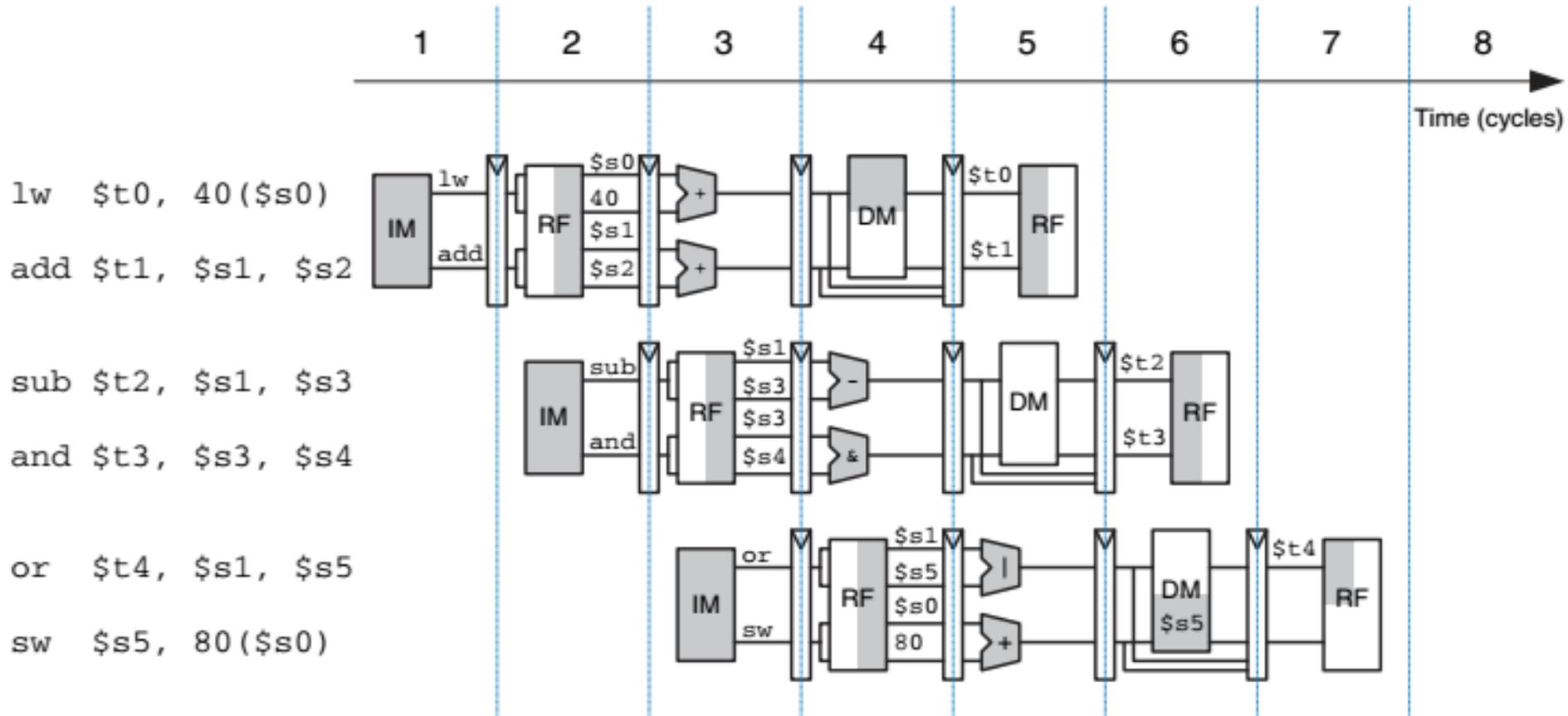
发射包

本例：发射宽度=2

单周期两条指令

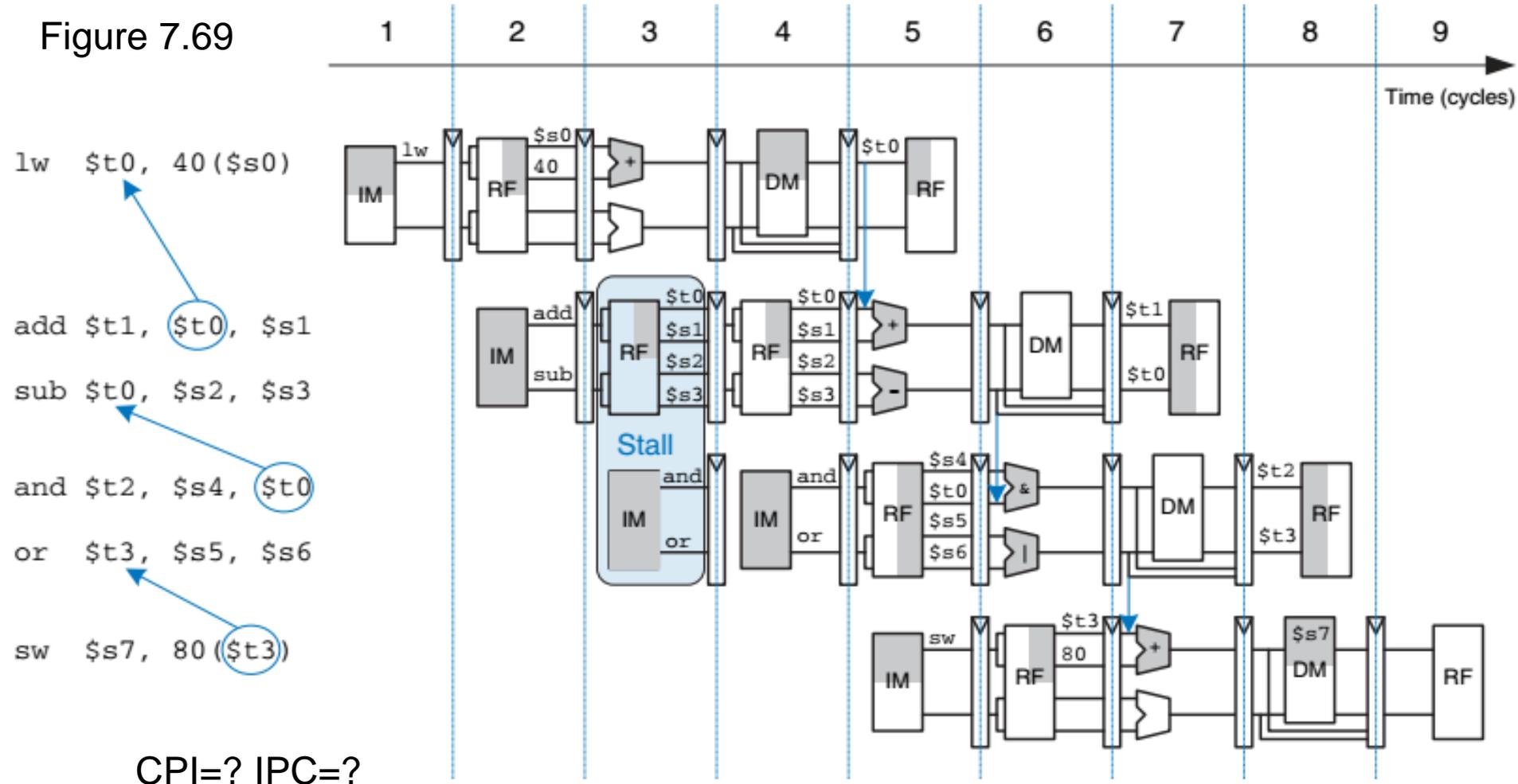
IF、ID为64位，多端口RF

A Two-Way Superscalar Pipeline

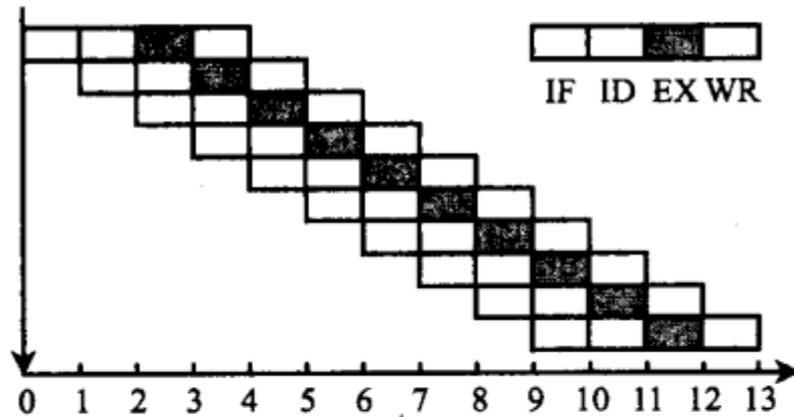


Program with data dependencies

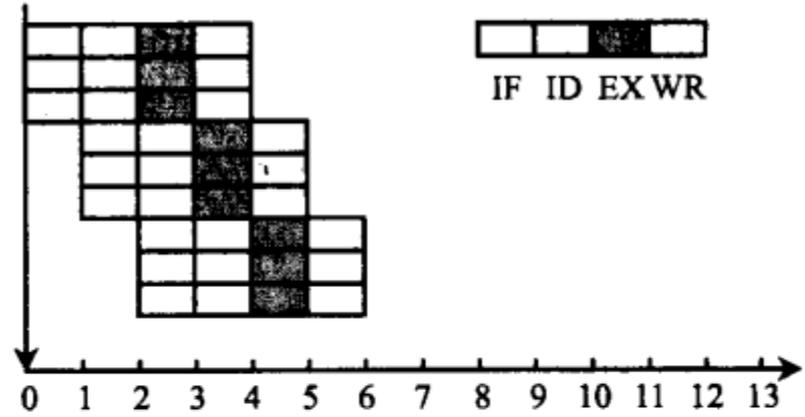
Figure 7.69



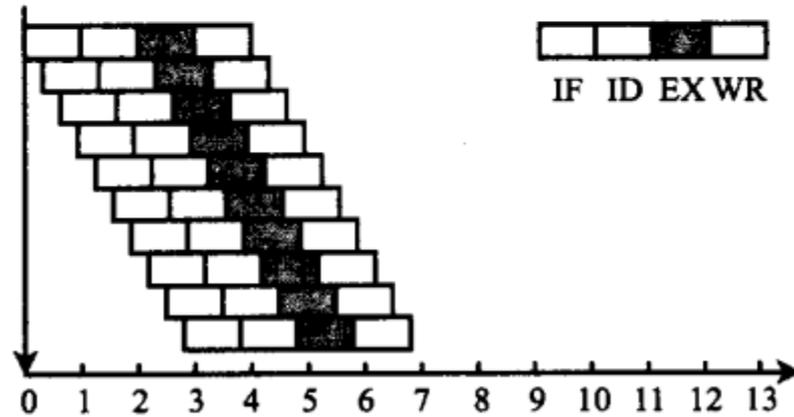
四种流水技术比较：图8.20@唐



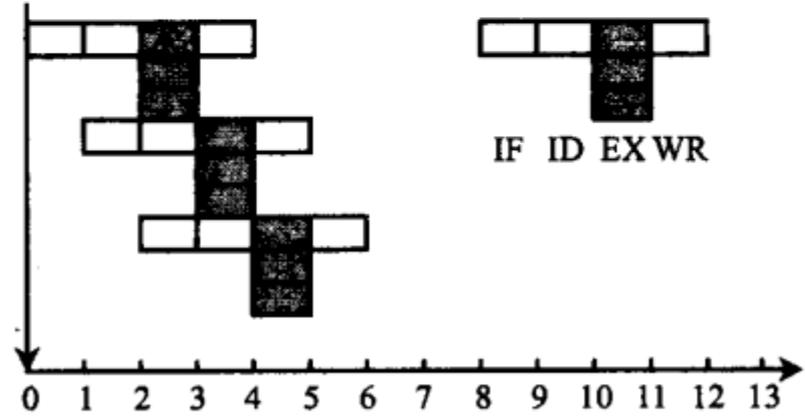
(a) 普通流水



(b) 超标量流水

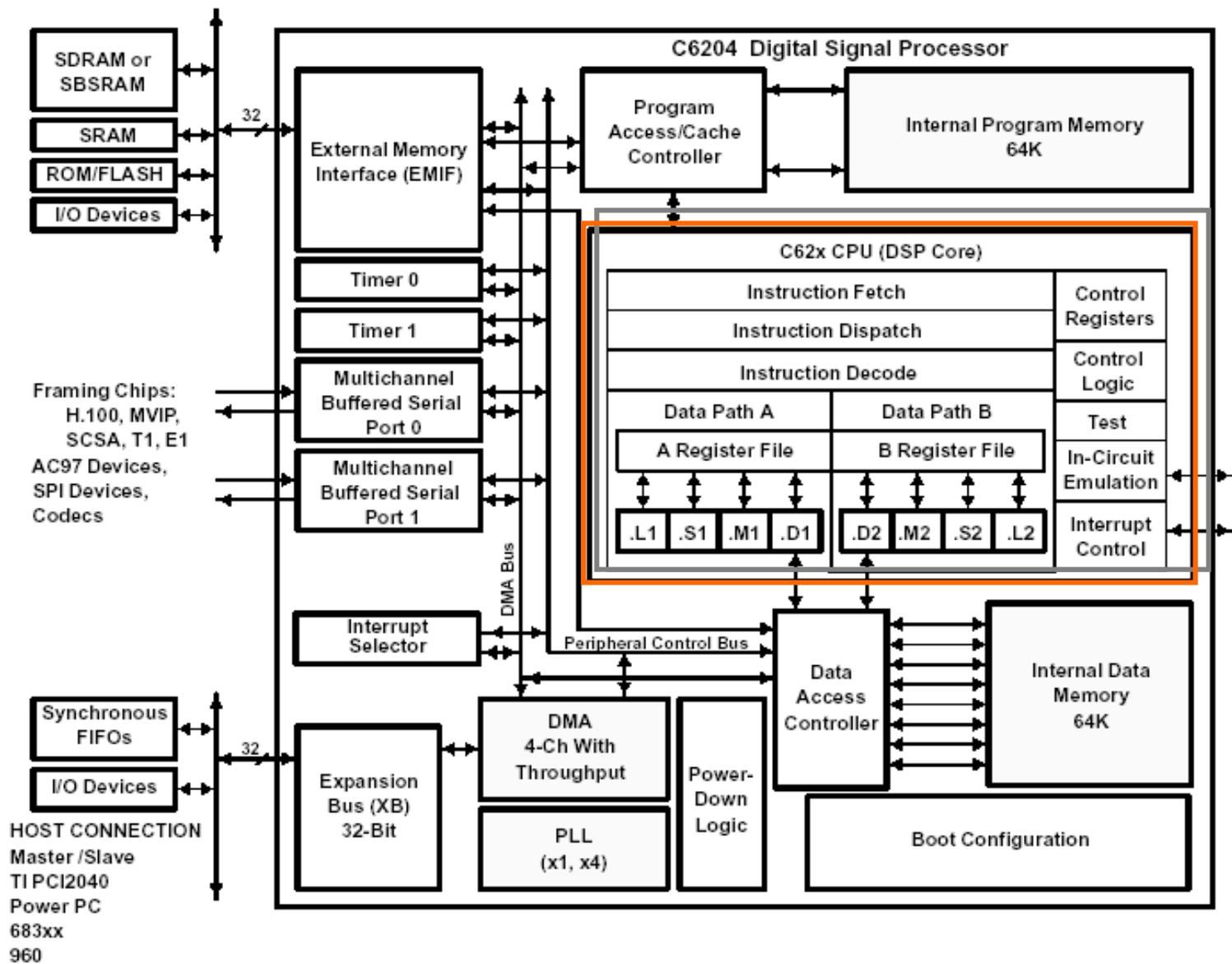


(c) 超流水线

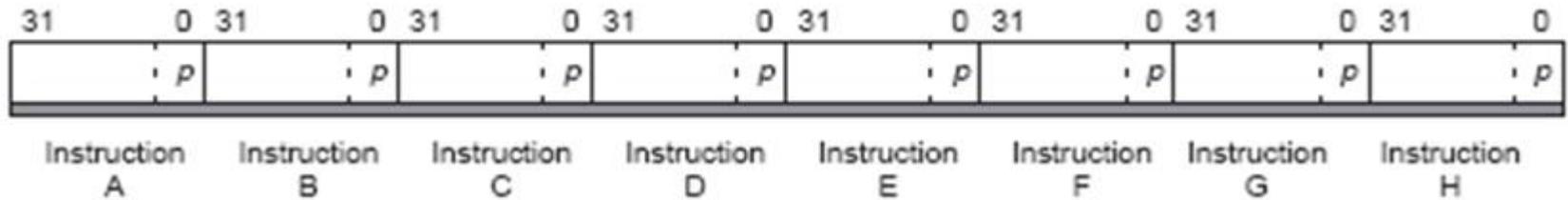


(d) 超长指令字

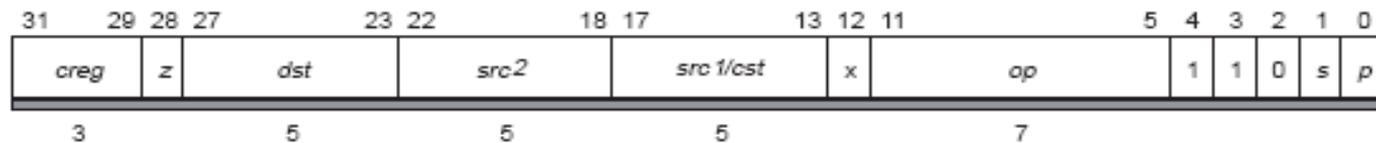
TMS320C64x: VLIW



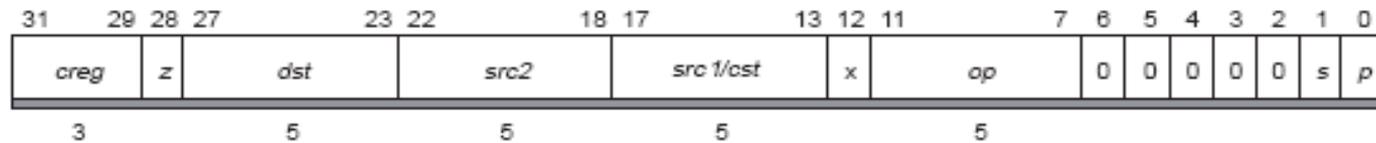
TMS320C64x指令字



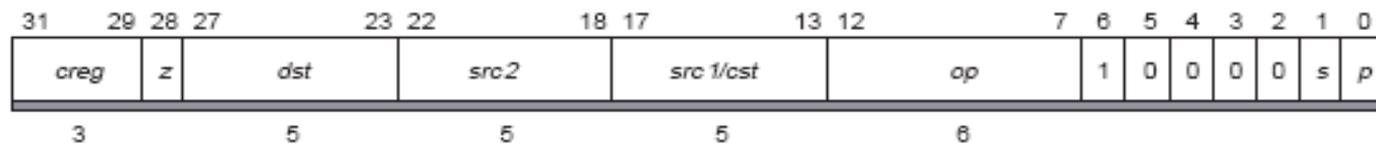
Operations on the .L unit



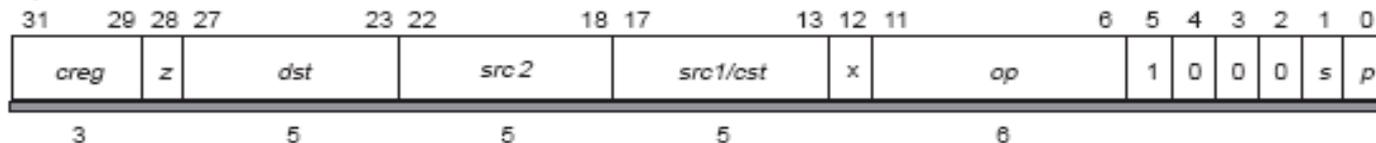
Operations on the .M unit



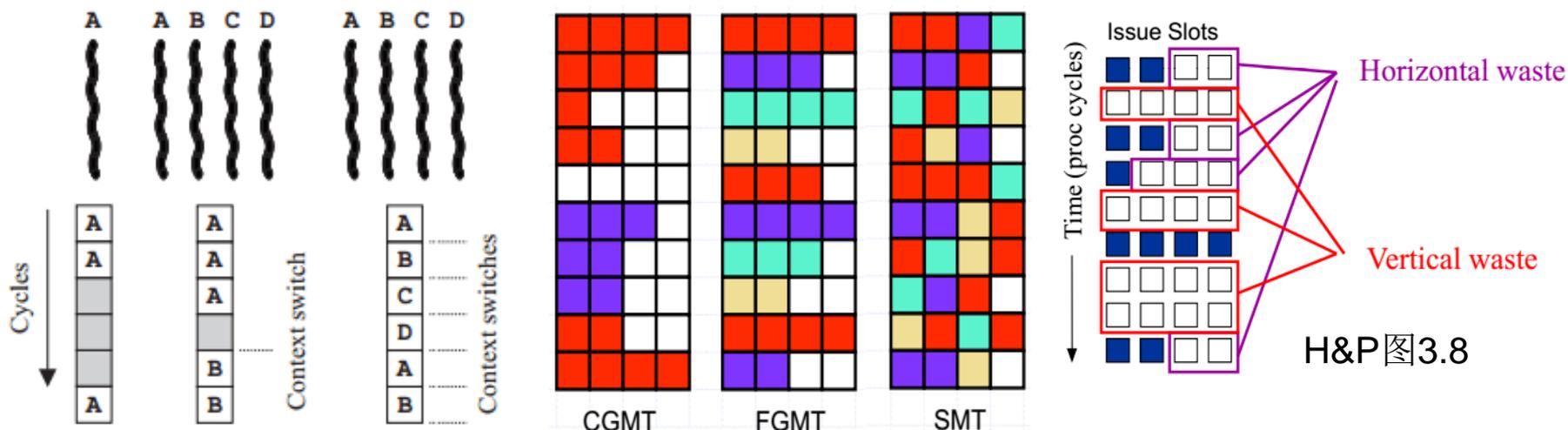
Operations on the .D unit



Operations on the .S unit

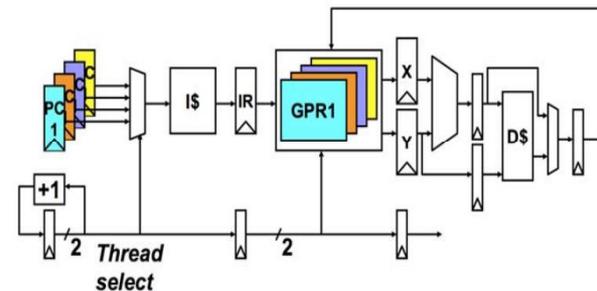
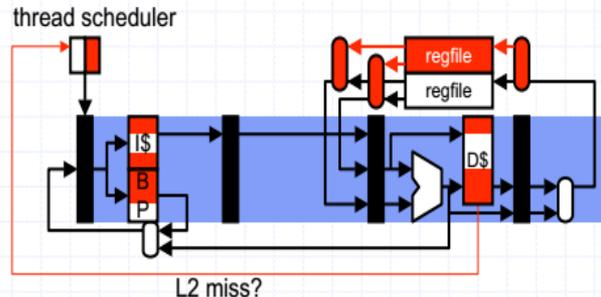
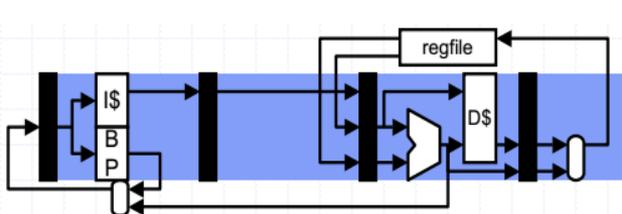


HMT技术（硬件调度）：TLP, MIMD, RV \$6.4

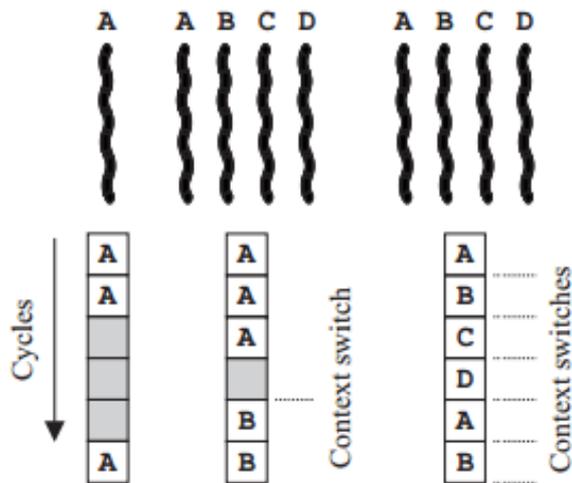


超标量流水线（4个ppl），4个线程，类似RV图6-5

- coarse-grain multithreading, Block Multithreading: 长时延事件=>线程切换



FGMT消除数据依赖和分支依赖



TID	Addr.	Inst.	Cycle								
			1	2	3	4	5	6	7	8	
0	0x00	BR 0x0C	F	D	E	M	W				
0	0x04	I		F	D	-	-	-			
0	0x08	I			F	-	-	-	-		
0	0x0C	I				F	D	E	M	W	

TID	Addr.	Inst.	Cycle								
			1	2	3	4	5	6	7	8	
0	0x00	BR 0x0C	F	D	E	M	W				
1	0x30	I		F	D	E	M	W			
2	0x60	I			F	D	E	M	W		
3	0x90	I				F	D	E	M	W	
0	0x0C	I					F	D	E	M	

- 细粒度多线程：提升了总吞吐量，但增加了单线程延迟

GPU体系结构，SIMD，MIMD，RV\$6.6

- DLP：数据级并行，由多个HMT SIMD处理器核组成MIMD处理器
 - 依靠HMT技术隐藏访存长时延，而非Cache（新版使用Cache）
 - 存储器面向带宽而非延迟，专用DRAM字宽更大

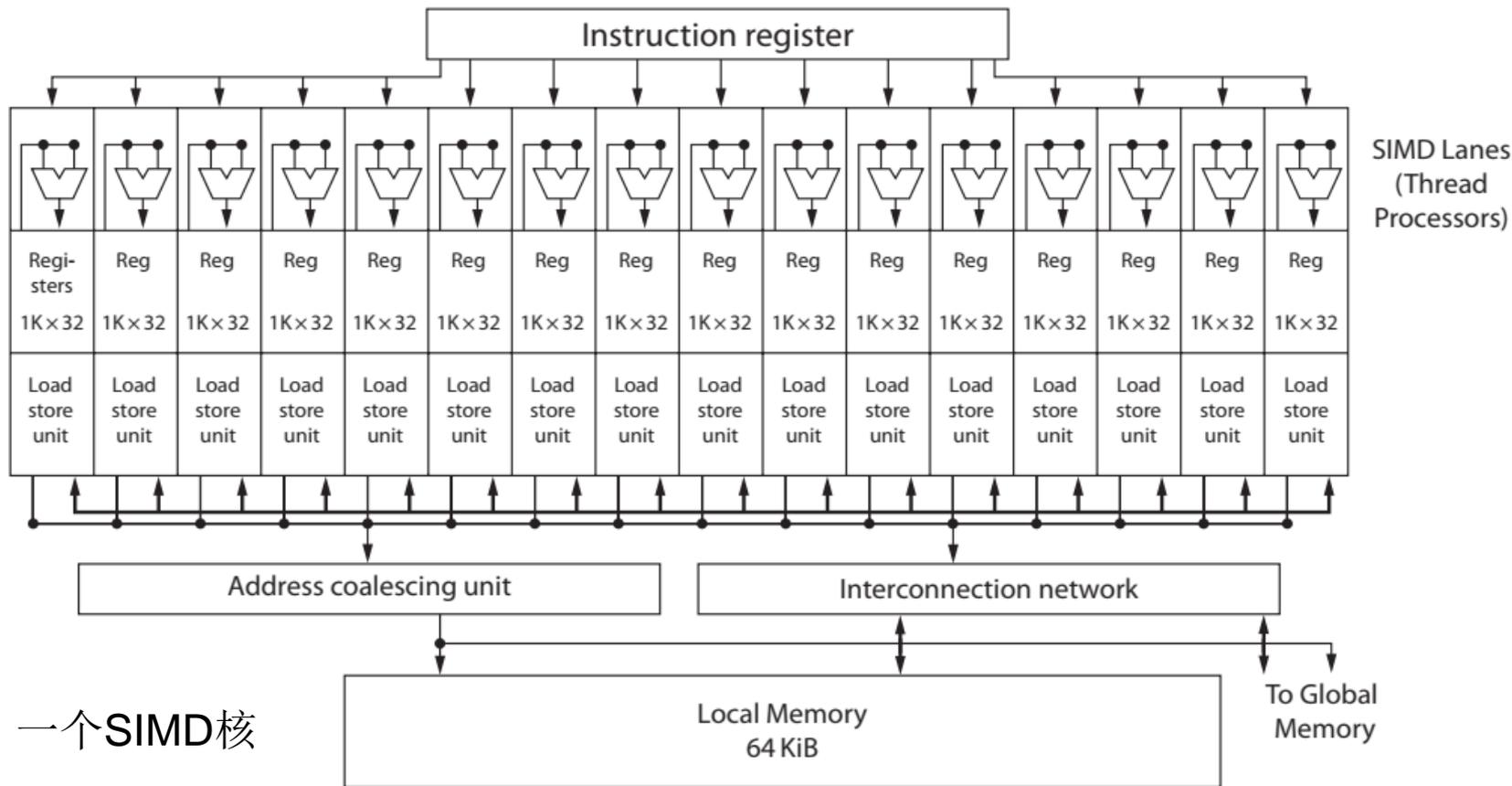
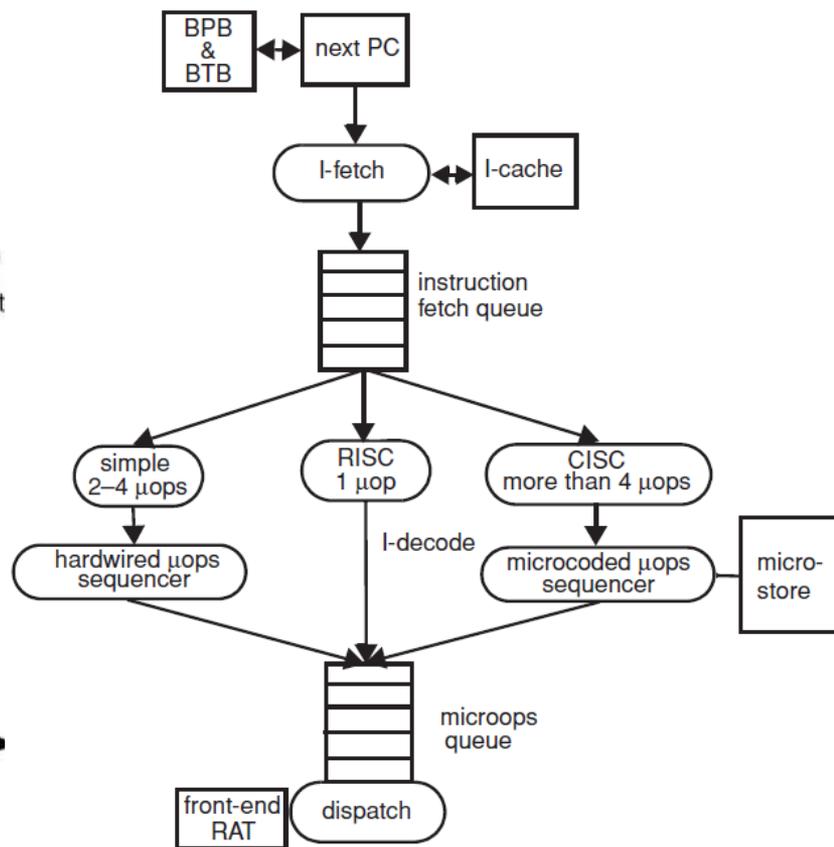
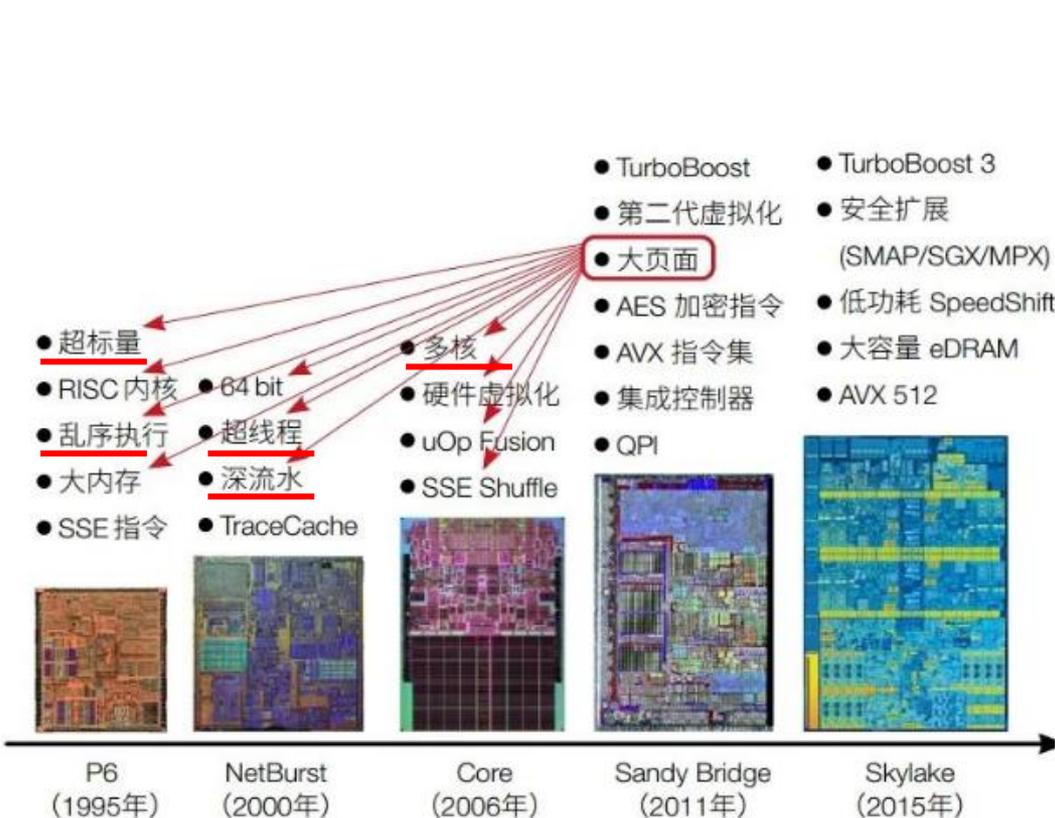


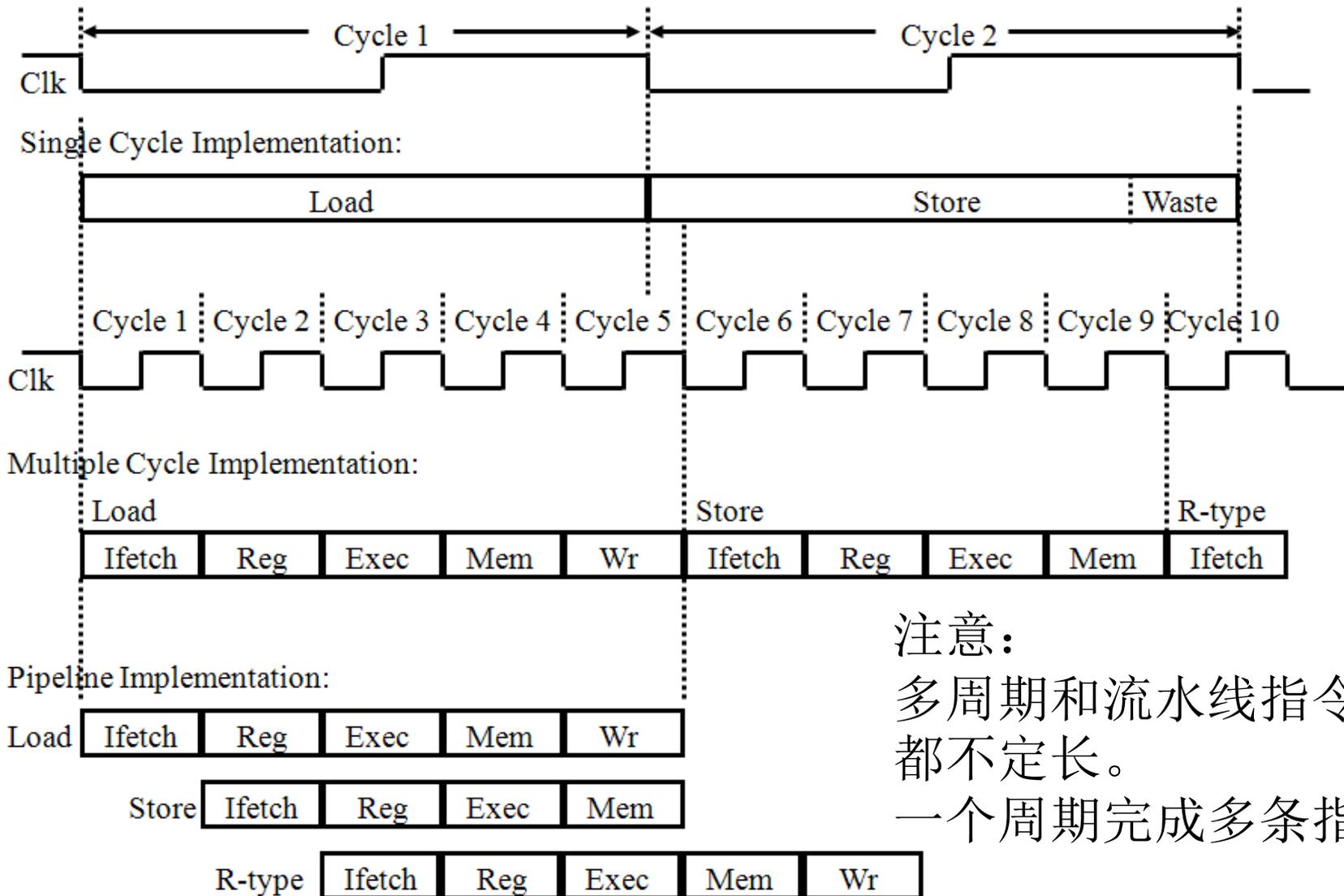
图6.9，一个SIMD核

Intel处理器架构演化



• 复杂ISA的前端

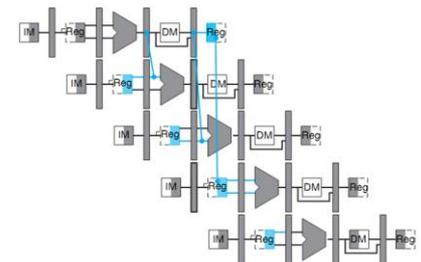
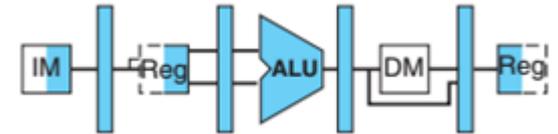
Single Cycle, Multiple Cycle, Pipeline



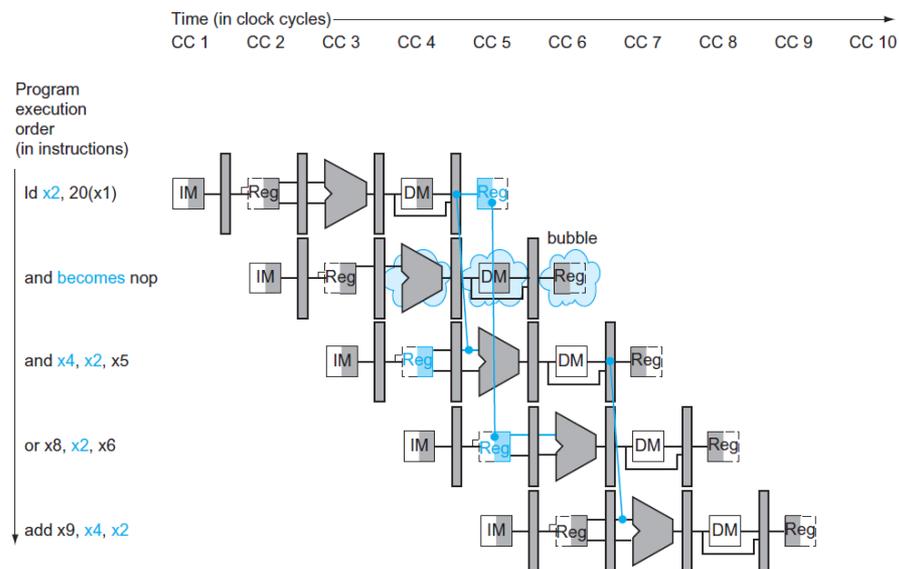
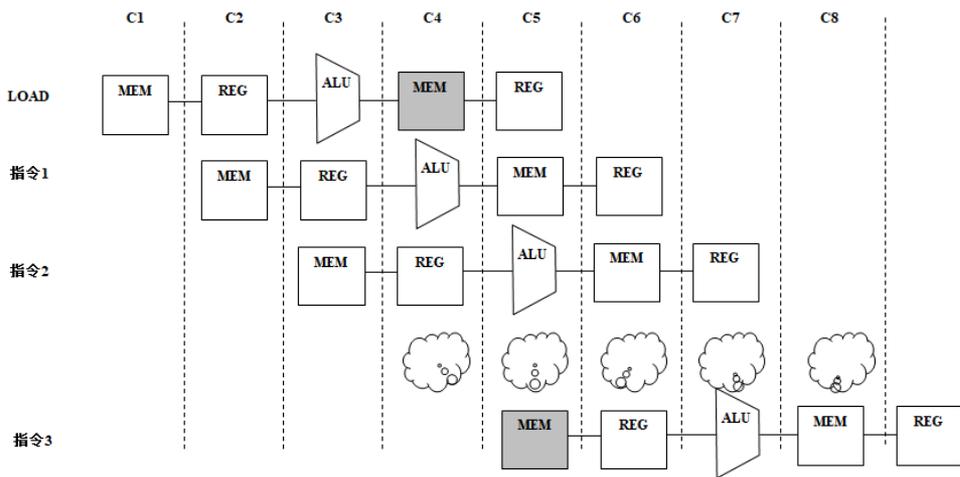
注意：
多周期和流水线指令周期
都不定长。
一个周期完成多条指令？

Pipelining Hazards: 类型, 原因, 处理

- “A **hazard** (冒险) is a **situation** that **prevents** starting the next instruction in **the next clock cycle**” ?
 - **Structural hazard**, 结构冲突
 - A required resource is busy (e.g. needed in multiple stages)
 - Can always solve a structural hazard by adding more hardware
 - **Memory Structural Hazards**: single port
 - separate instruction/data memories & load/store
 - at most one memory access per instruction
 - **RegFile Structural Hazards**: single write-port
 - load-ALU: ALU instruction **passing** MEM
 - **Data hazard**, 数据依赖
 - Data dependency between instructions : 生产者消费者
 - Need to **wait** for previous instruction to complete its data **write**
 - Forwarding: grab operand from Interstage-Buf, rather than RegFile.
 - **Multilevel Bypass**: EX bypassing, Load/Store Bypassing, beq bypassing
 - **Can't** solve all cases with forwarding: load-use-data need to Stall
 - WB stage: RegFile **Double Pumping**——“内推”
 - **Control hazard**, 转移损失
 - **Flow** of execution depends on previous instruction
 - Flushing: **Single** cycle branch
 - Branch prediction: **Zero** cycle branch——IF段



stall语义与实现：保证执行正确



• Stall点

- 结构冲突：IF段stall
 - MEM, RegFile, **ID**
- RAW依赖：ID段stall
 - Load-used
- 分支依赖：IF段stall

• Interlocking规则

- Load-used: 单拍
- RAW: 多拍
- beq: 多拍

RAW: forwarding to ?

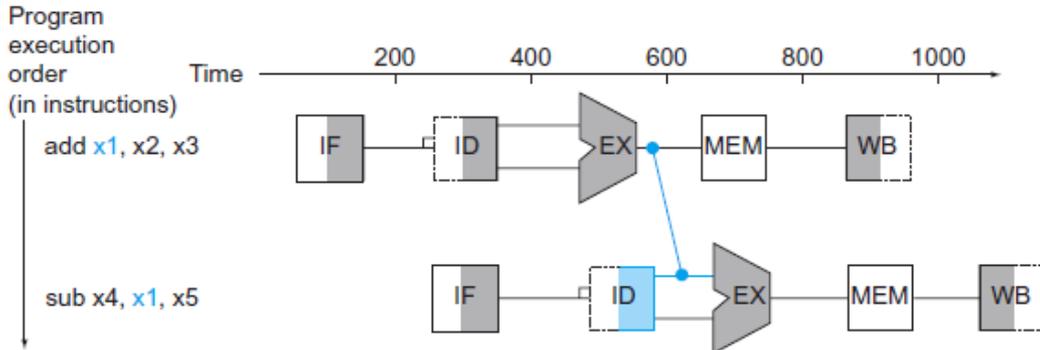


图4-27: EX/MEM forwarding to EX

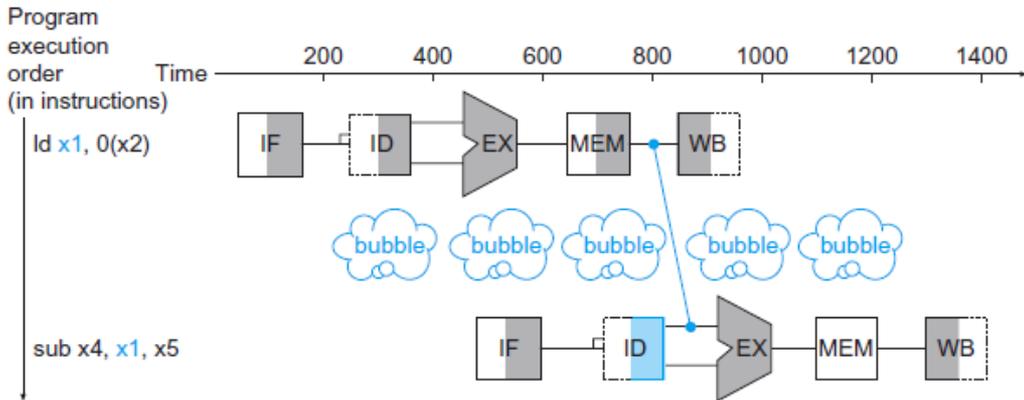
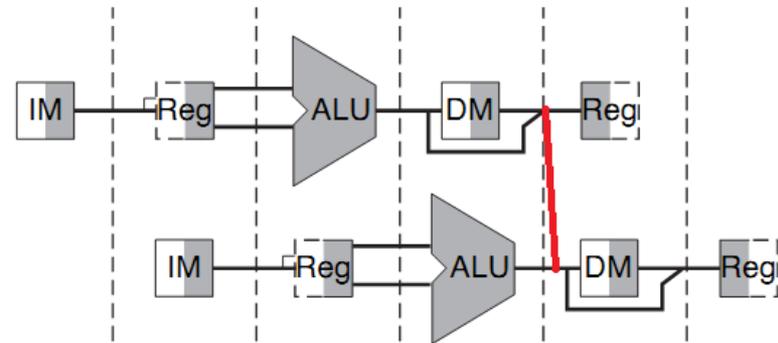


图4-28: stall, MEM/WB forwarding to EX

数据通路

前推: 哪儿冒险就推到哪儿
从哪儿前推?

FW控制器



lw-sw: MEM/WB forwarding to MEM

执行时间？

- add x2, x1, x1
- add x3, x2, x2
- ld x4, x3, 0; *ld x4, 0(x3)*
- add x5, x4, x4

- 无冒险处理部件
- 有冒险处理部件（FW, interlock）

小结，作业

- 分支冒险
 - 三种处理方法
 - Stall
 - 分支预测：not taken, taken
 - 延迟分支
 - Flush: “clear up, bubble down”
 - 单周期beq
 - RAW
- 典型的流水线的多发射技术有哪些？
- 作业：4.25

Thank You