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：4.5*
- *RV的五级流水线实现：4.6*
- *Hazard问题：4.5.2*
  - *结构冲突：哈佛结构*
  - *数据依赖：4.7*
    - *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：4.8
    - 编译技术：延迟分支
    - 硬件优化：提前完成，投机，预测
- 多发射技术：4.10
- 硬件多线程：6.4

beq taken？

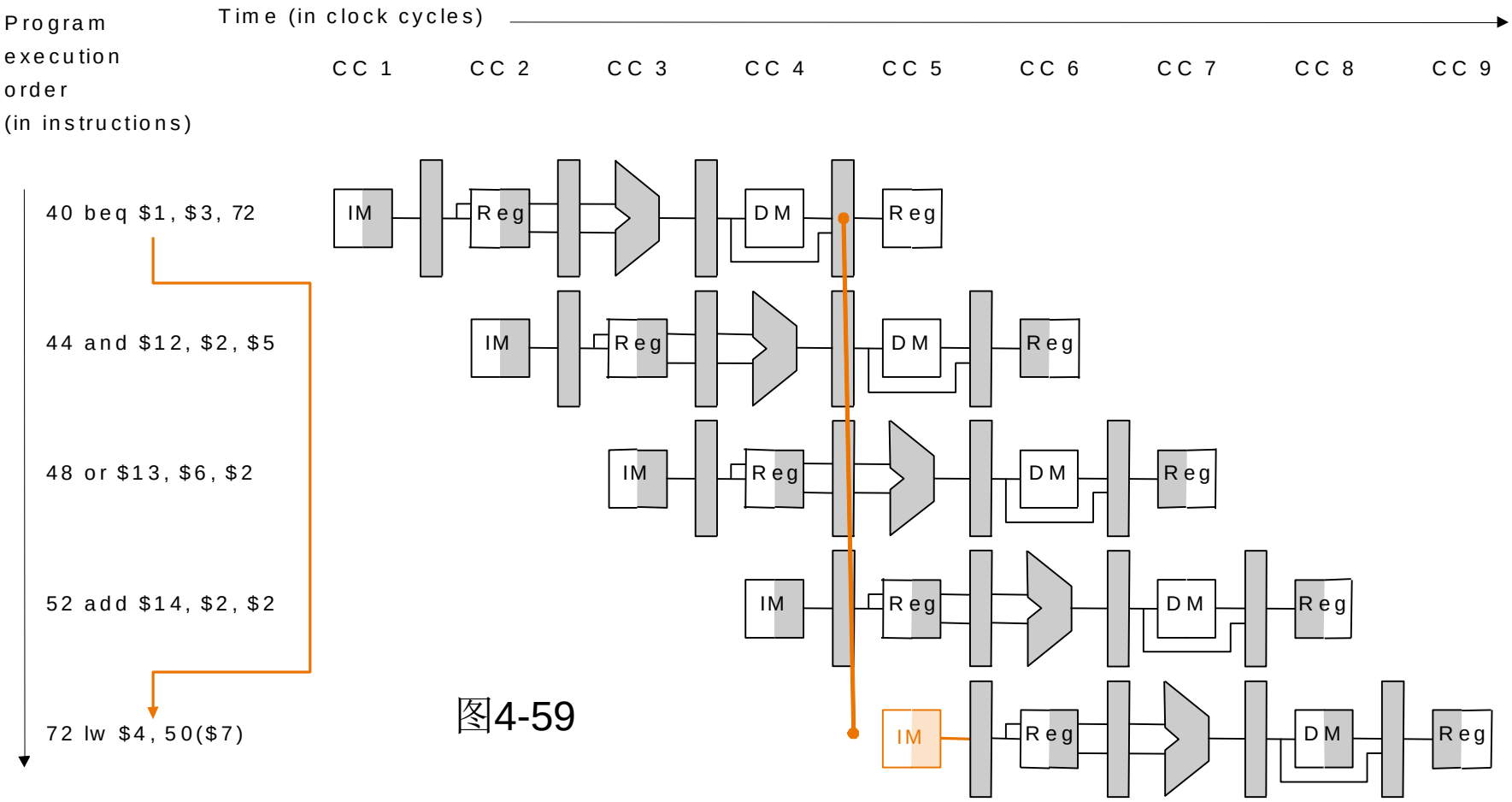
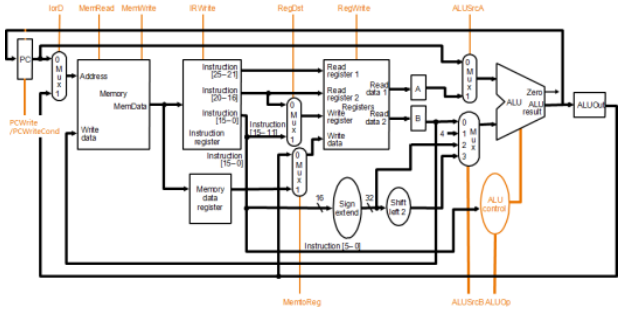


图4-59

beq指令完成时间？

- 以哪一条指令为基址？
- 能否在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		



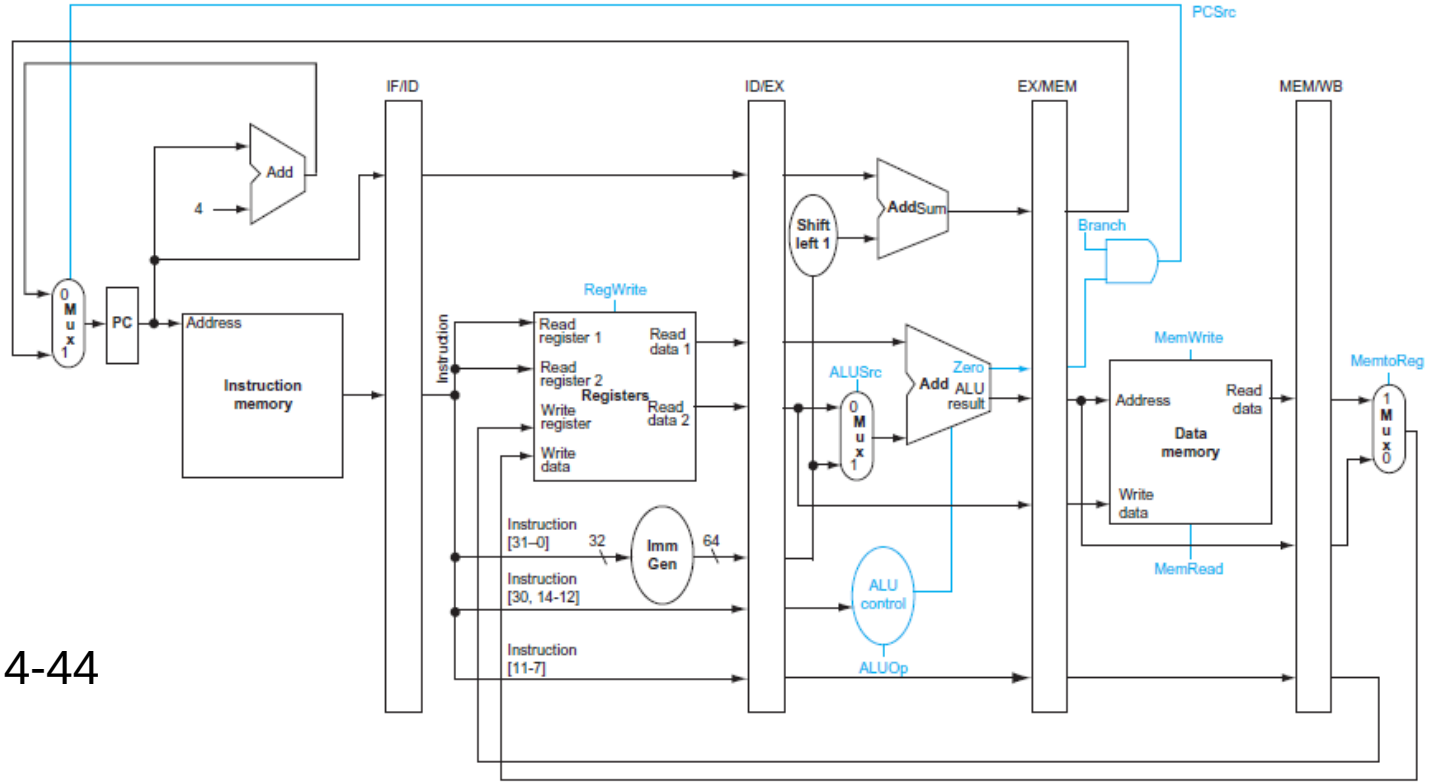


图4-44

beq branch hazard：stall？

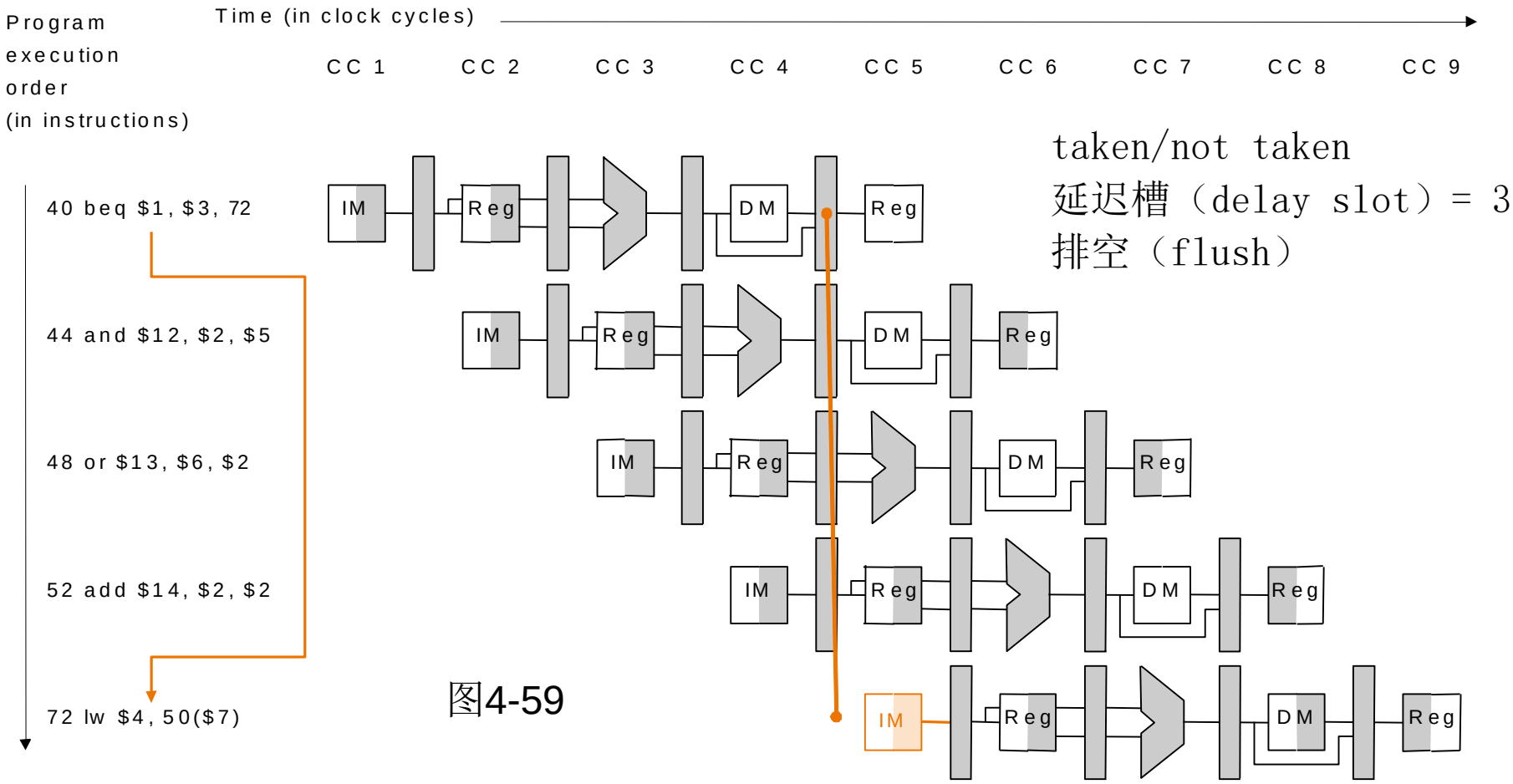


图4-59

“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！

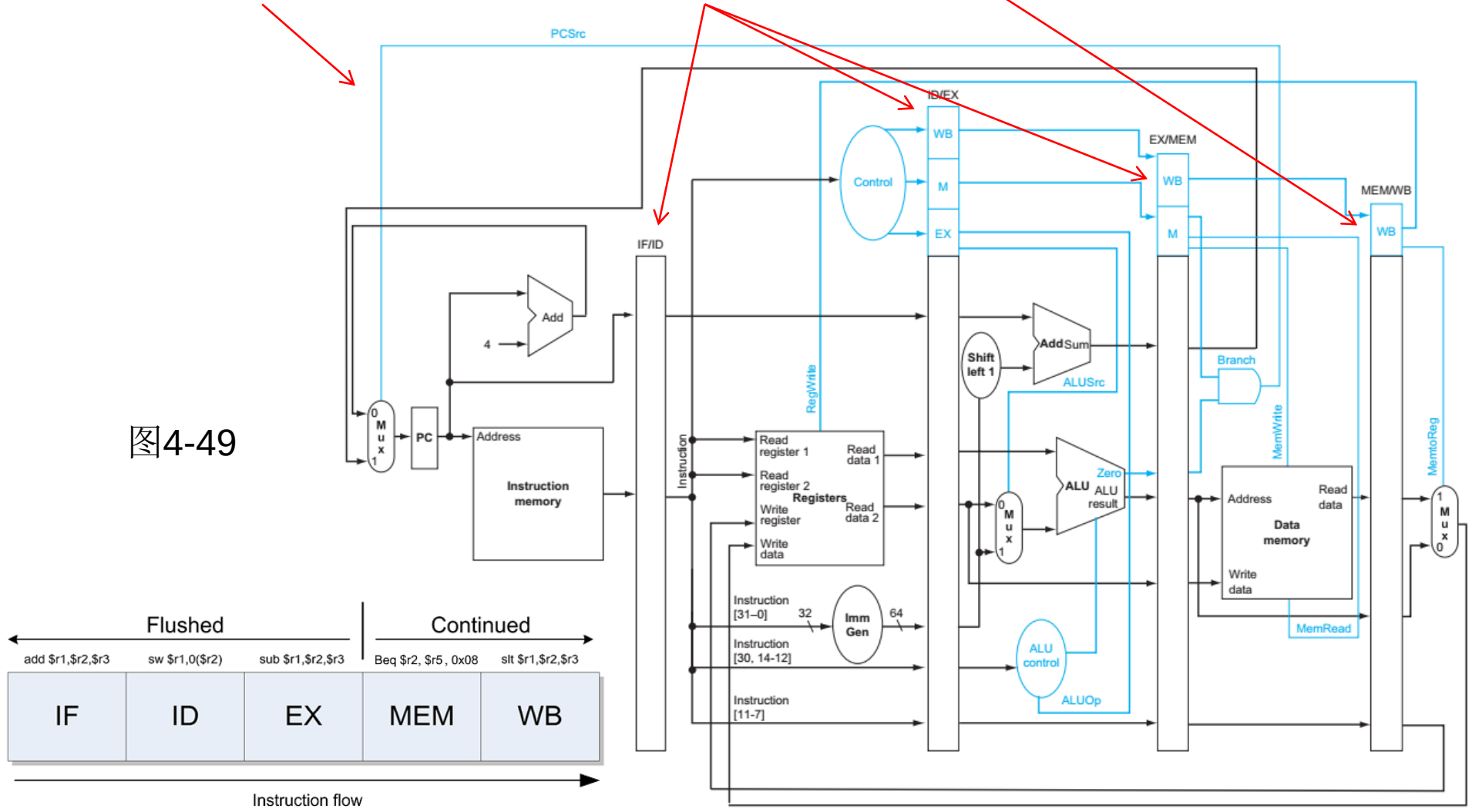
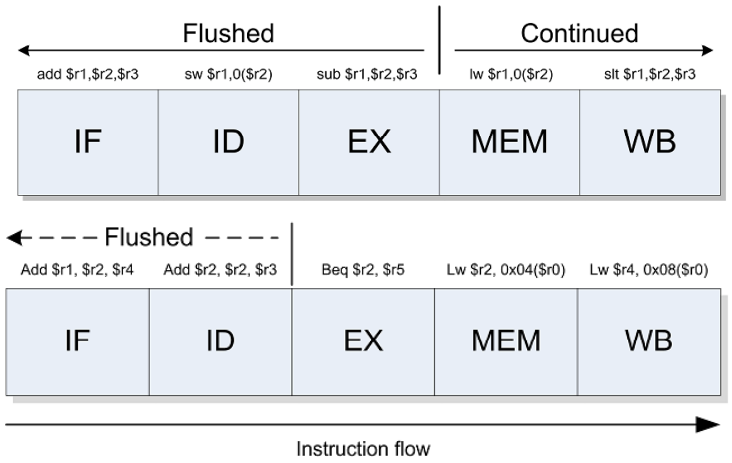
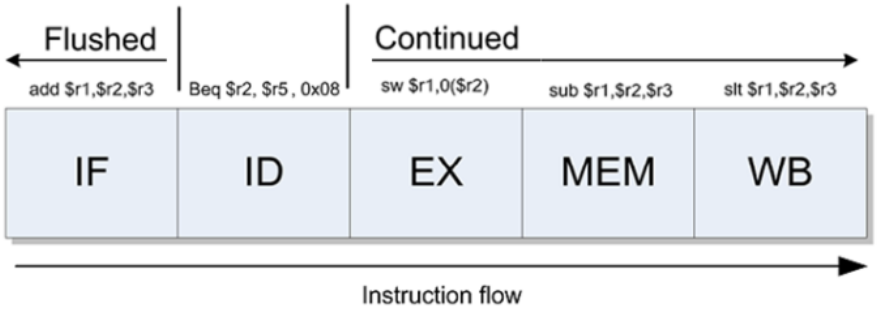


图4-49

减小beq损失：单周期分支，ID段完成





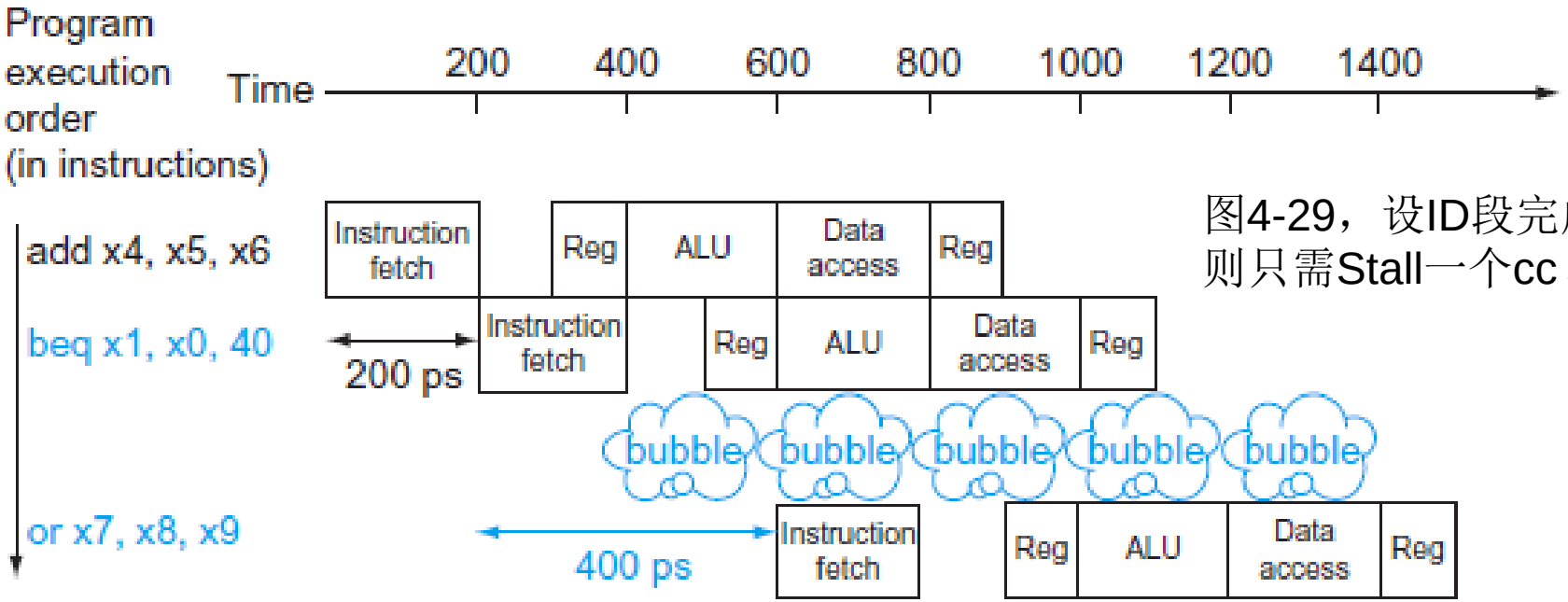
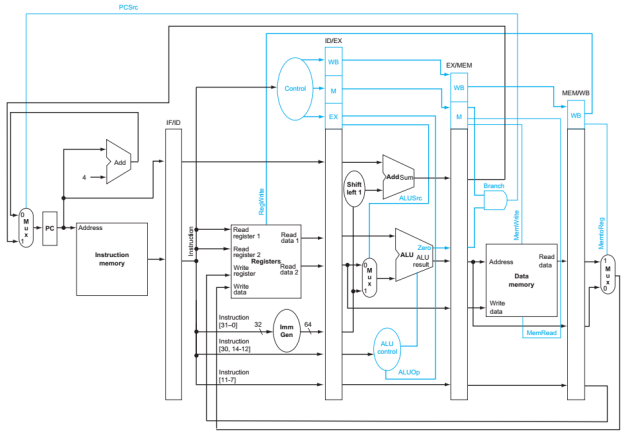
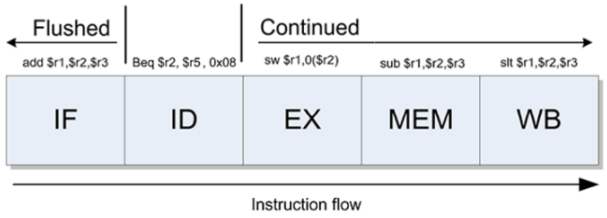


图4-29，设ID段完成，则只需Stall一个cc！

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（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4-62*）
    - *“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



beq实现：单周期延迟

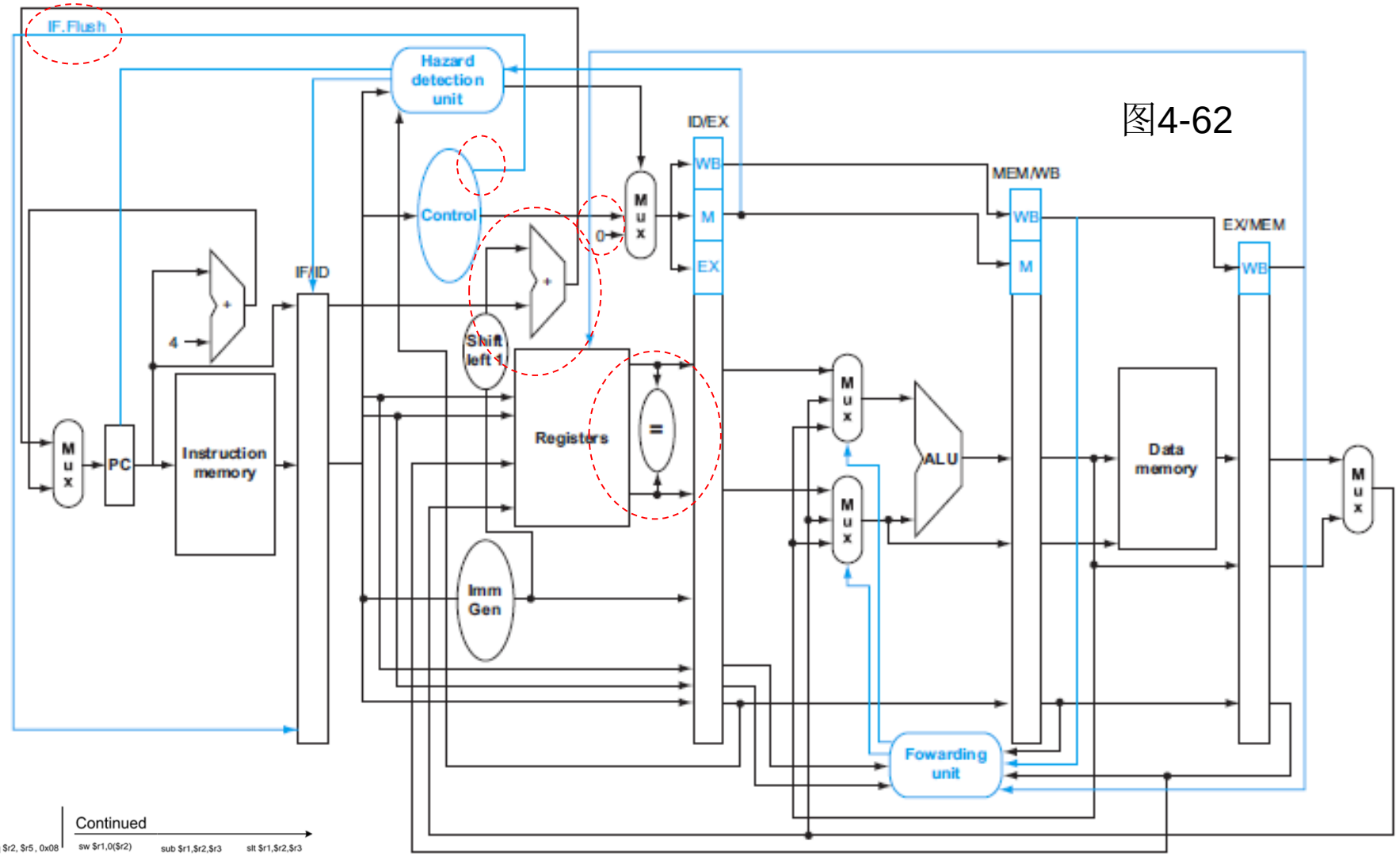
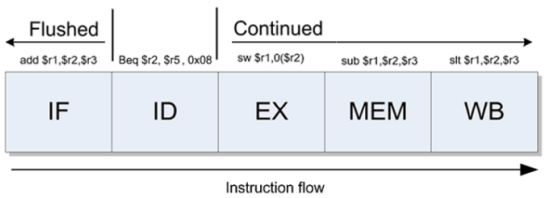


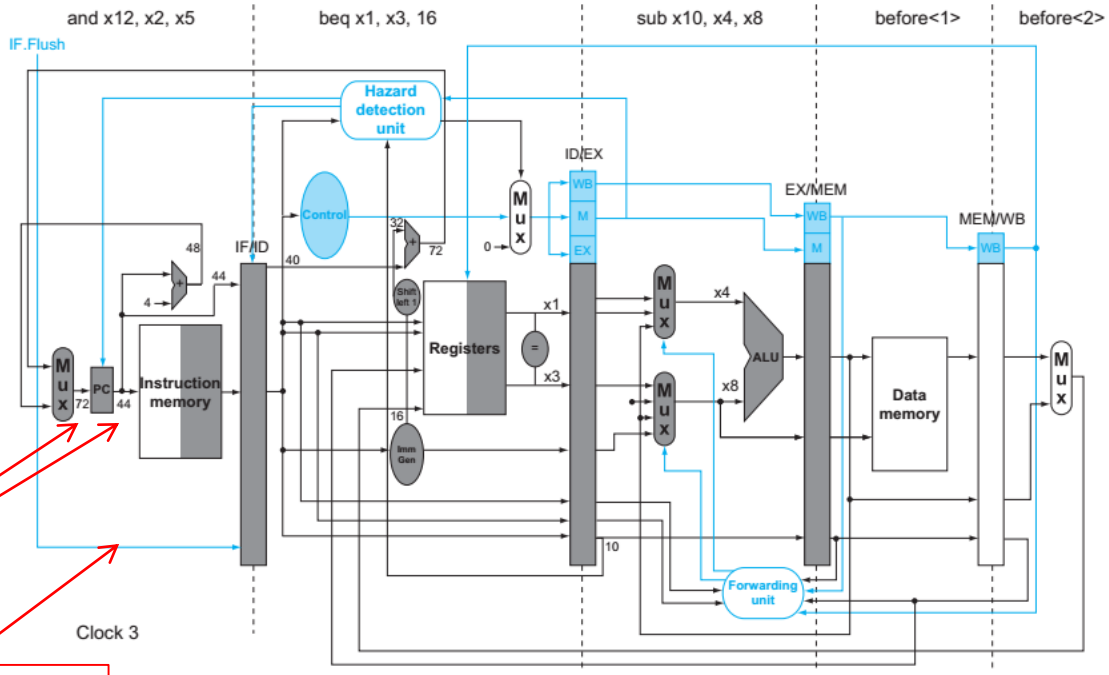
图4-62



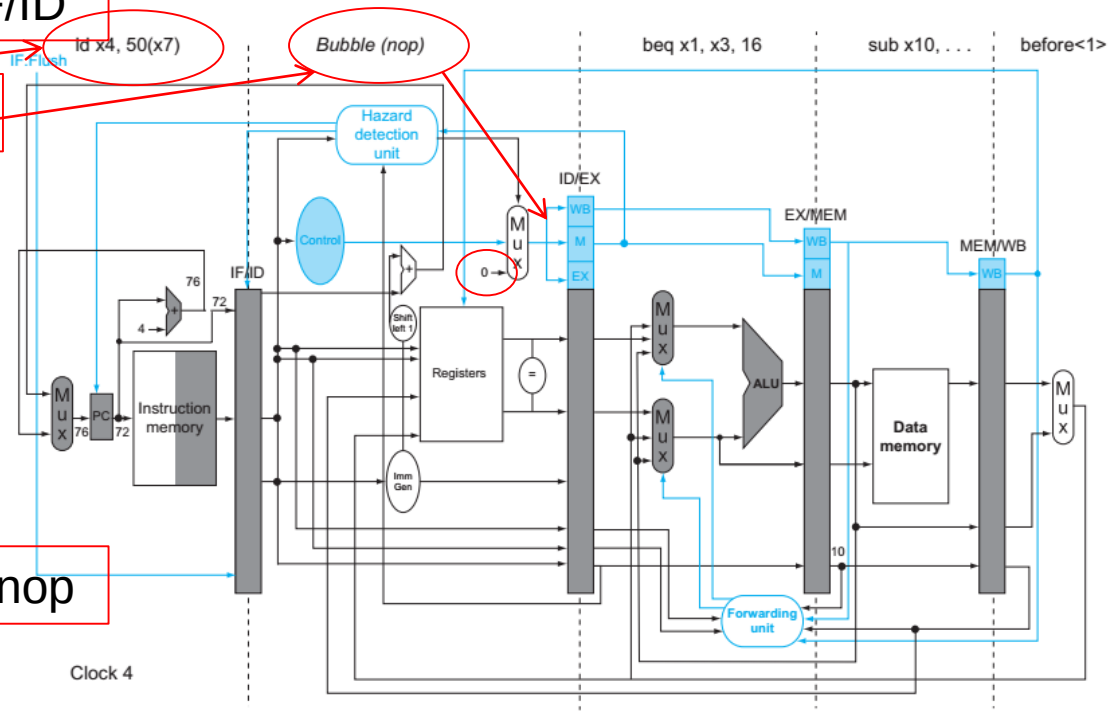
Delay slot：one cycle!

Flush & Taken

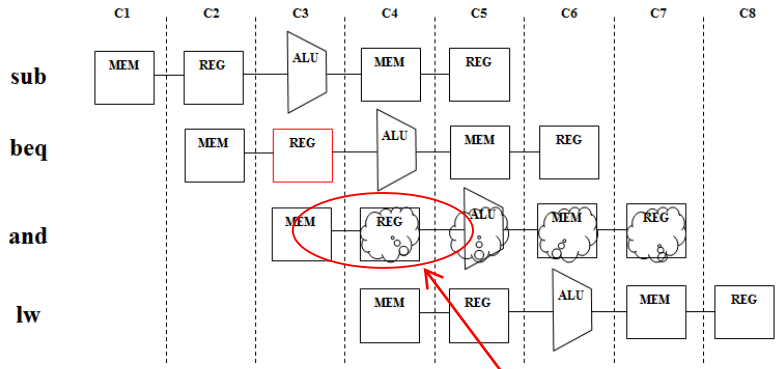
```
36  sub  x10, x4, x8
40  beq  x1,  x3, 16
44  and  x12, x2, x5
48  or   x13, x2, x6
52  add  x14, x4, x2
56  sub  x15, x6, x7
. . .
72  ld   x4, 50(x7)
```



判定taken，则：更新PC，Clear IF/ID



取“ld”，bubble down



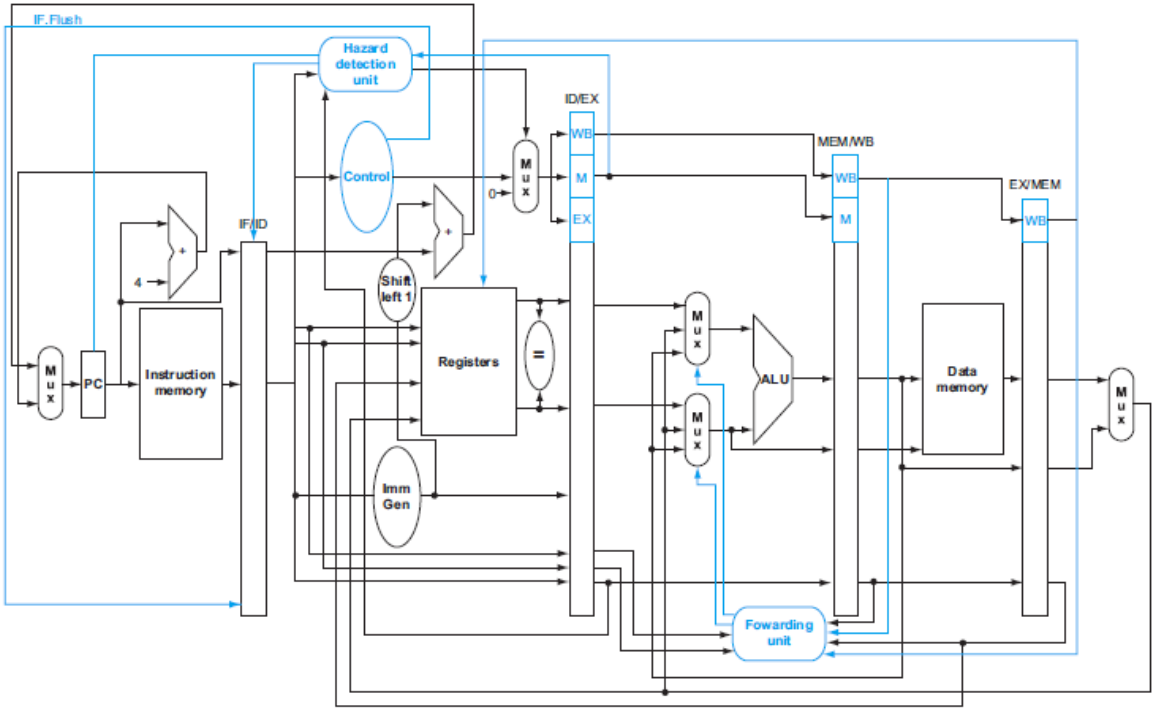
从cc4开始变为nop

- 图4-60

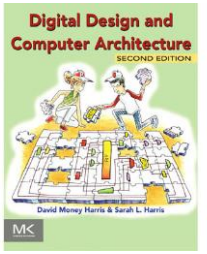
单周期beq的RAW，$4.8.2

- 例一：
  add $4，$5，$6
  beq $4，$2，40
  beq $3，$4，50
  。 。 。

- 例二：
  lw $4，20($1)
  beq $4，$2，40
  。 。 。

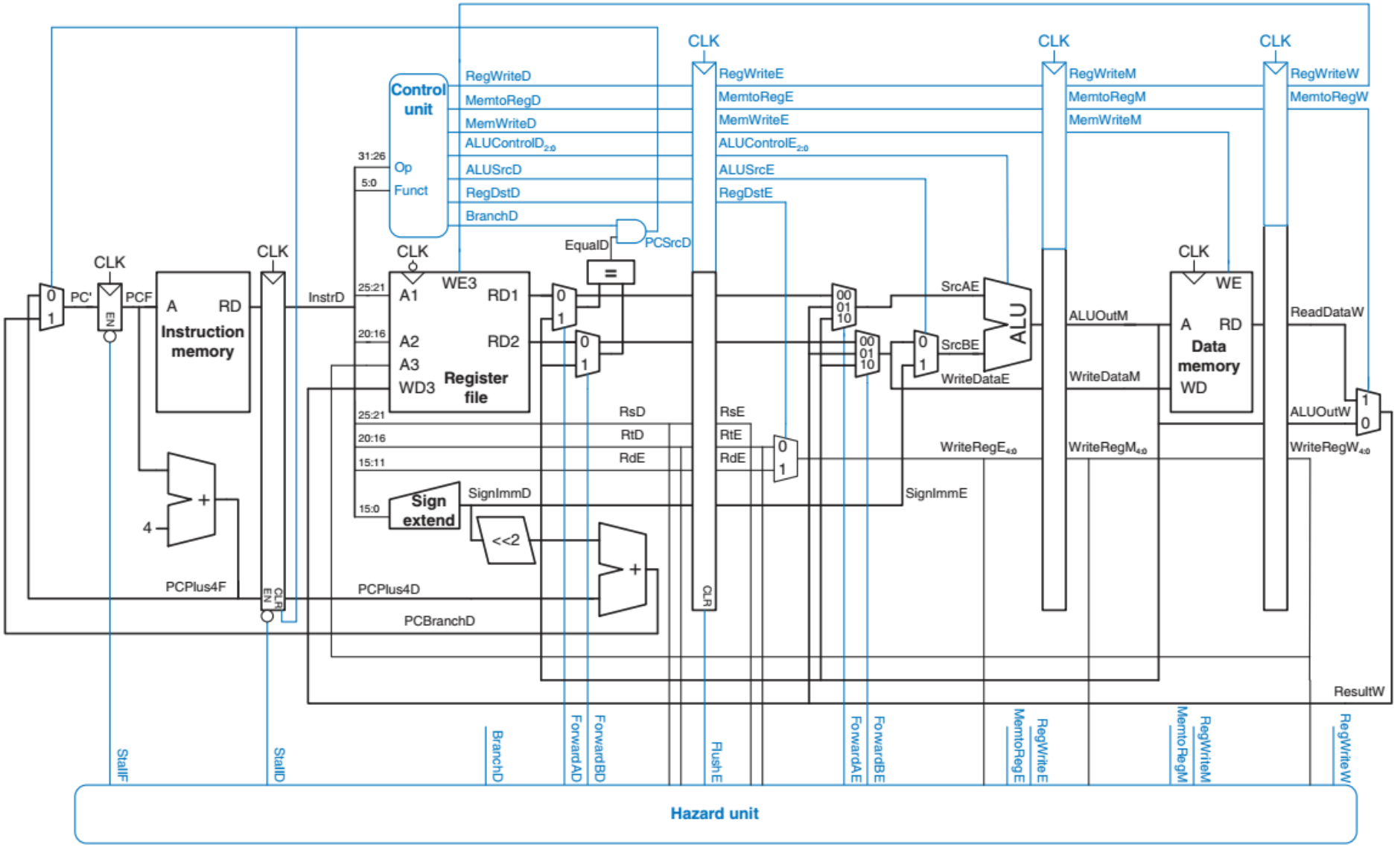


- 时空图？
- 需要新的interlock规则和ID段forwarding规则？



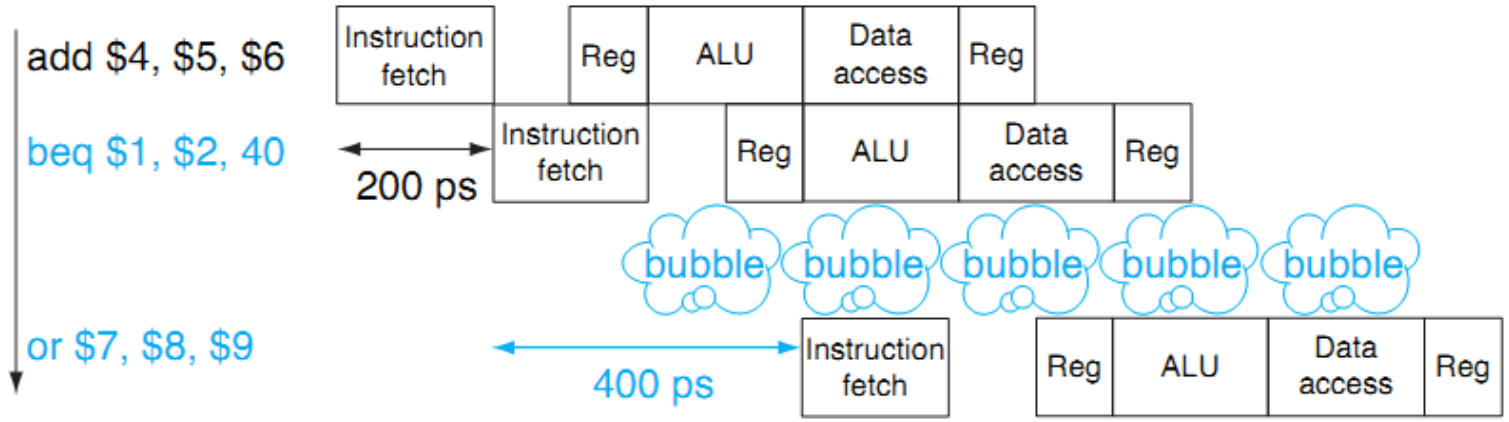
with full hazard handling

图7.58



beq指令的性能，$4.5

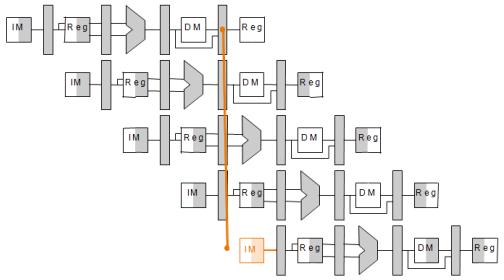
- 单周期分支stall对性能的影响，例
  - SPECINT2006中，条件分支占17%。设其他指令CPI=1。
  - 单周期条件分支需stall一个周期，因此平均CPI=1.17，即性能下降1.17倍。
  - 原始版本（beq在MEM完成）的CPI=？



分支冒险的处理技术：三类

- RV分支指令占比：36%（图2-41）
- 方法1 Stall：$4.5.2，$4.8.2
- 方法2 分支预测：发生？目标地址？硬件，$4.8.3
  - 静态预测：假设taken/not taken，投机执行（Speculation）
  - 动态预测：程序运行时使用实时信息进行预测，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，$4.5.2
  - 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

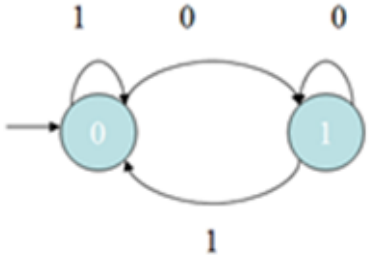
分支预测：静态预测，投机执行



- 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

- 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/直接映射Cache：预测目标地址
    - 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改
  - 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

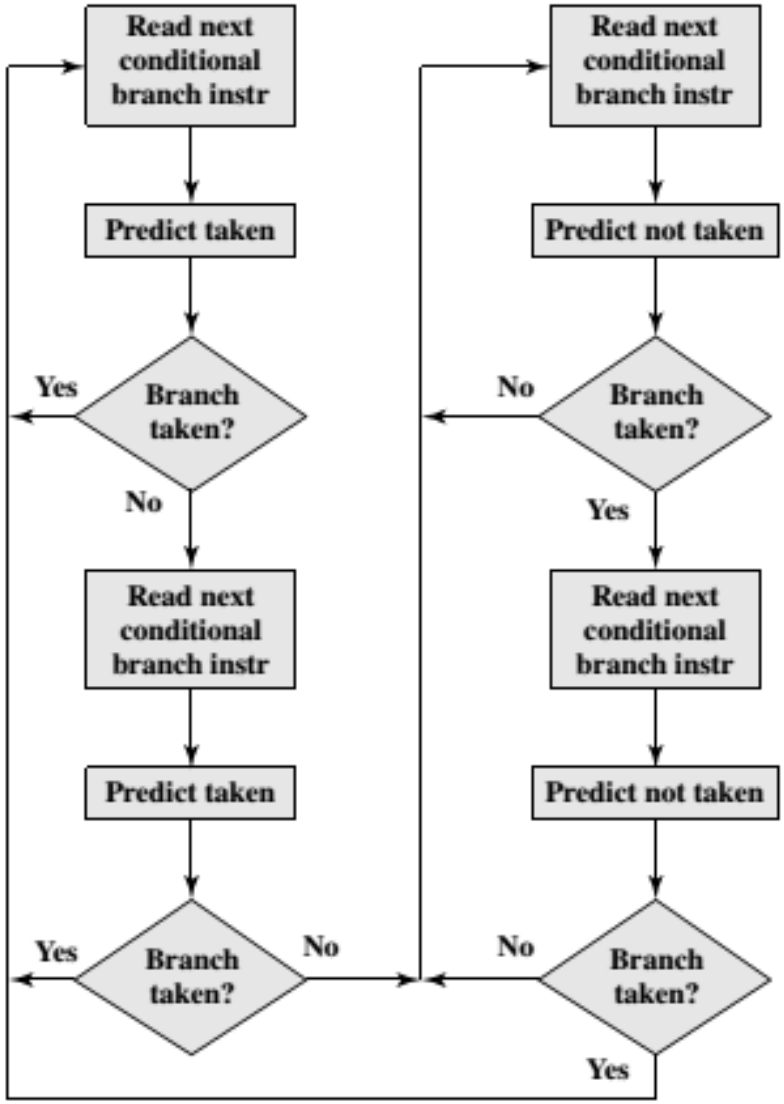
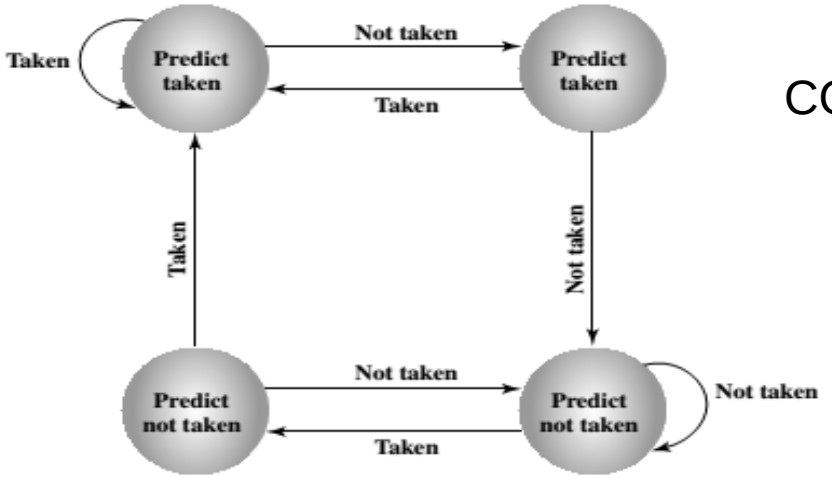


Figure 12.18 Branch Prediction Flowchart

COA

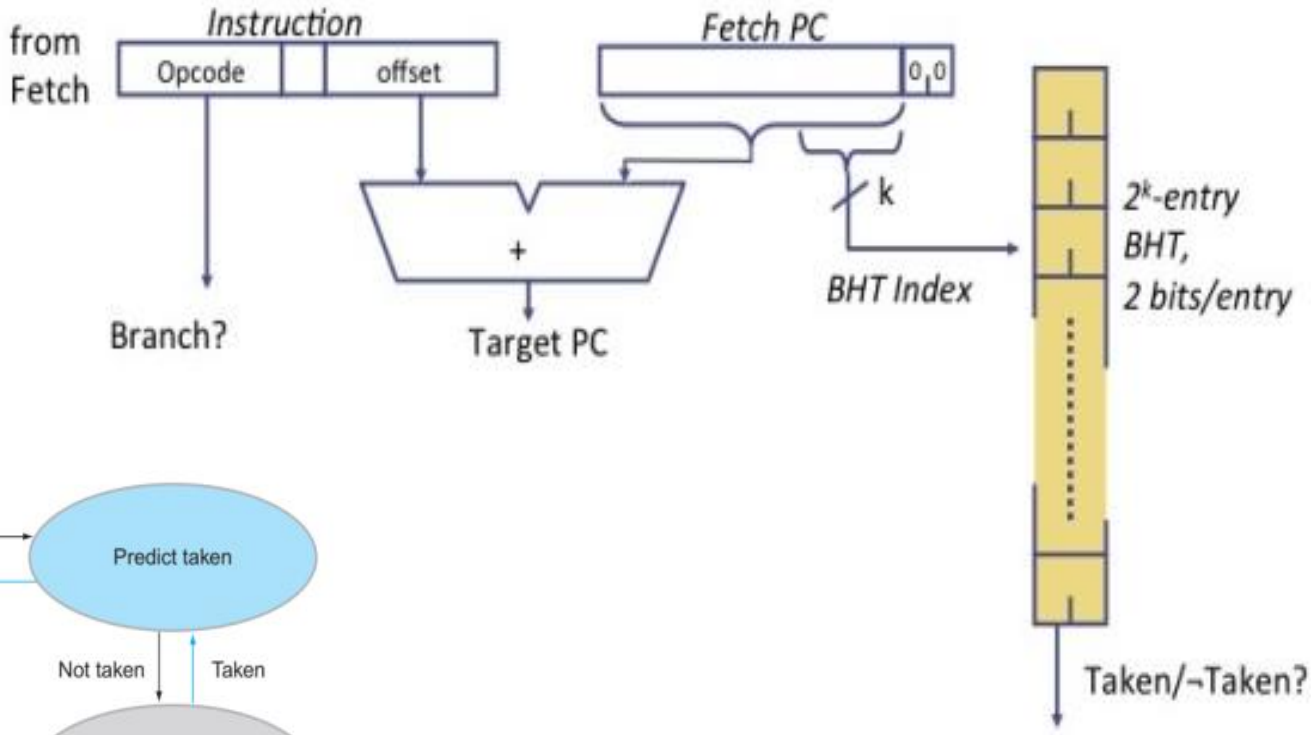


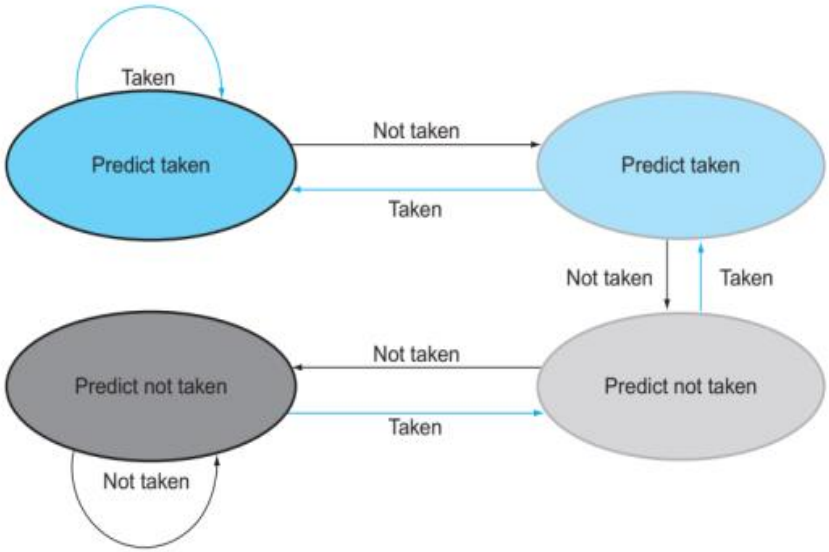
COA

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。

BHT（branch history table）

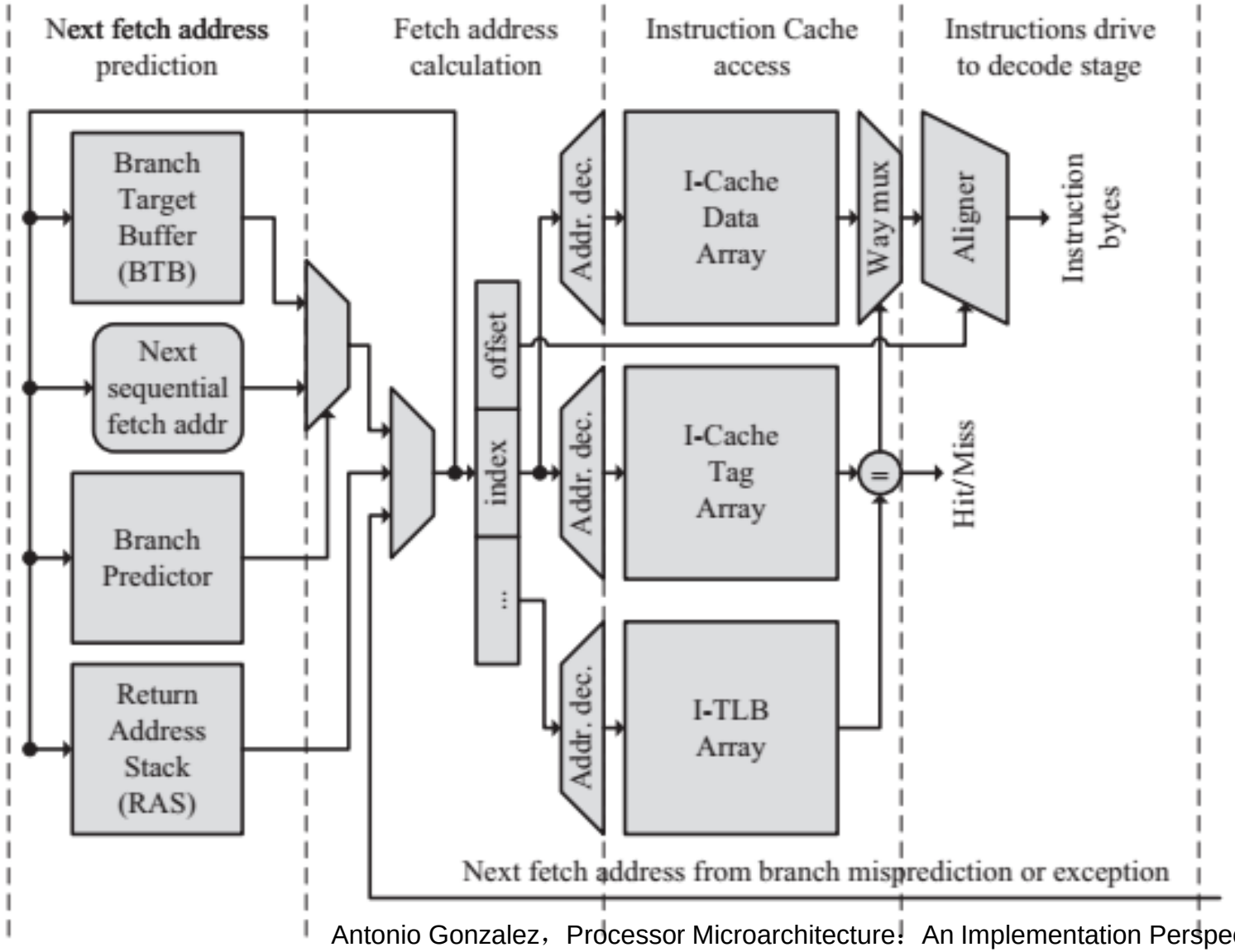
RV图4-61





- 以PC的低k位为索引

Branch Prediction实现



Antonio Gonzalez，Processor Microarchitecture：An Implementation Perspective，2010

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)
```

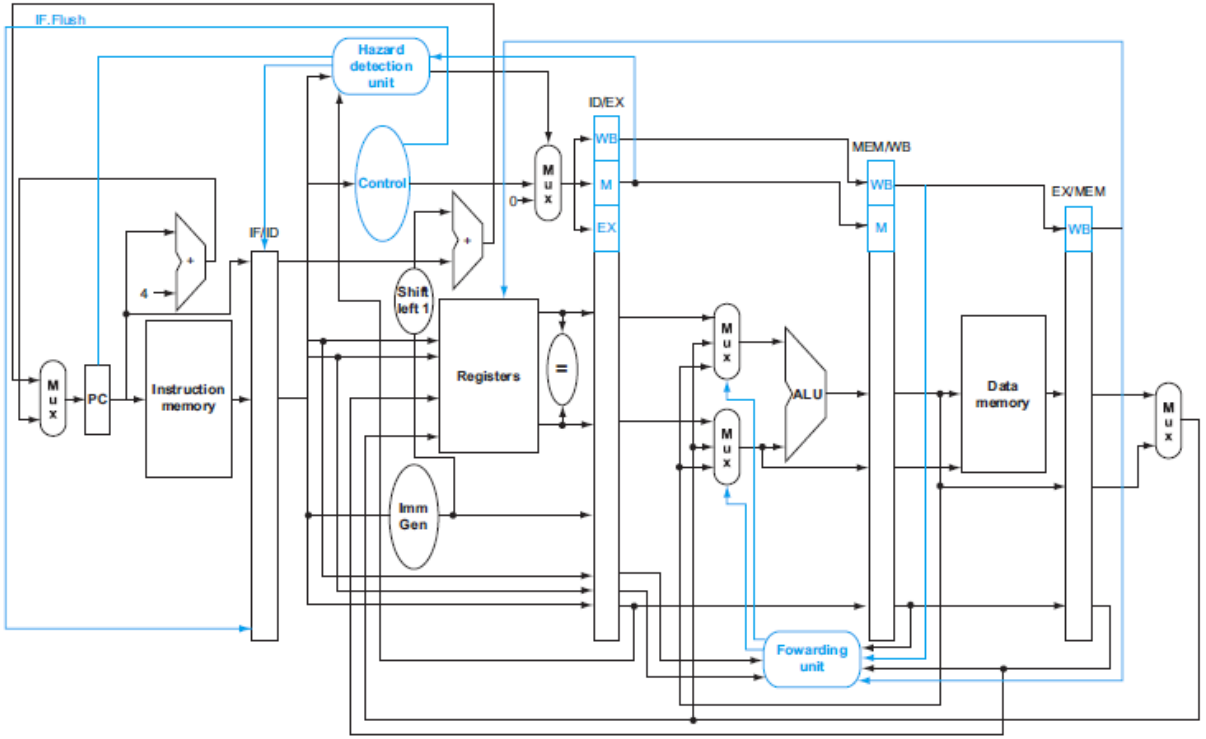
```
T = 该分支被选中
N = 该分支没有被选择

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 ... NNNNNNNTTTTTTTTT ... TTTTTTTTT  (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

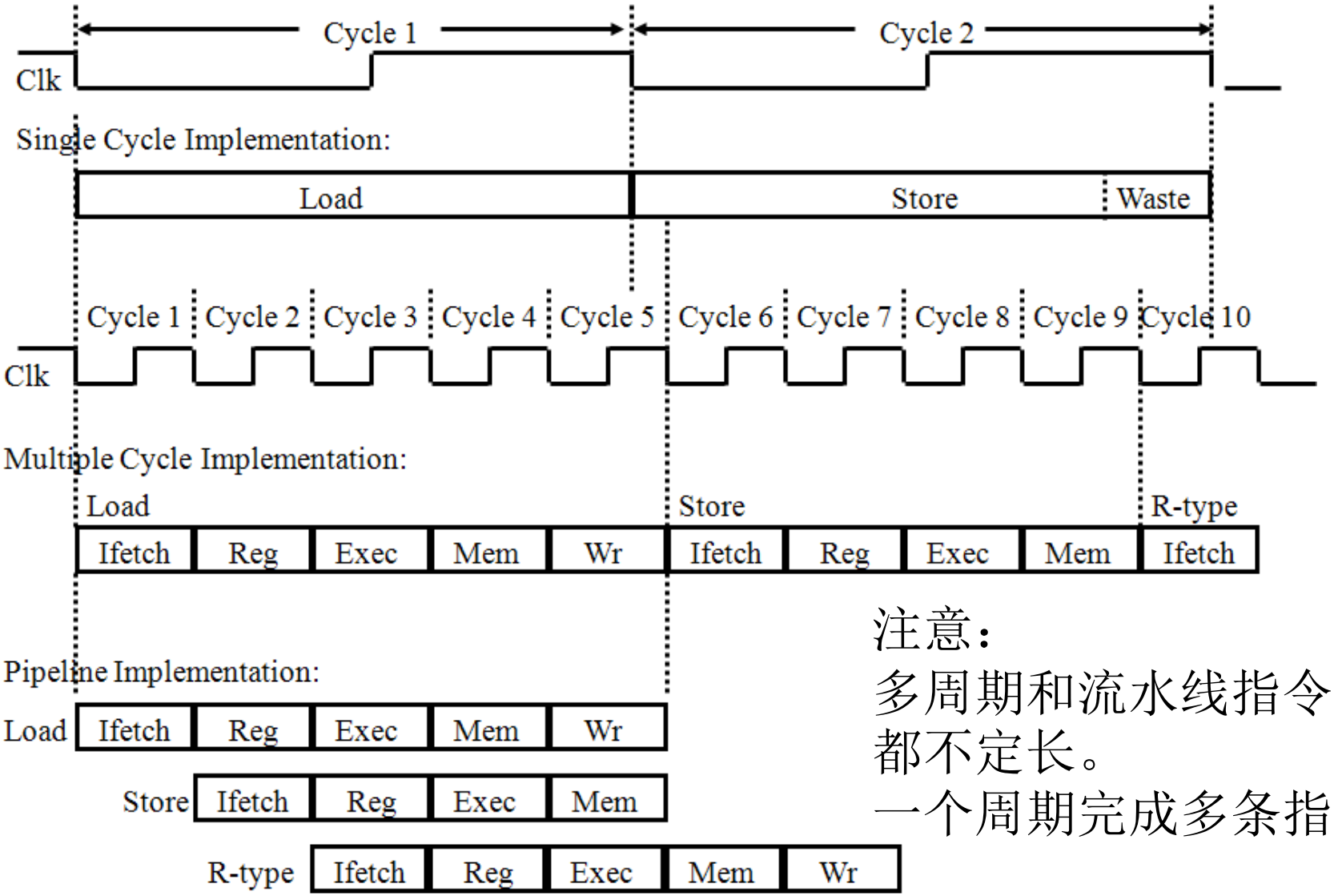
无条件分支？

- 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



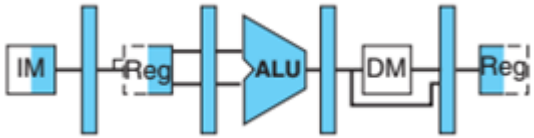
Single Cycle, Multiple Cycle, Pipeline



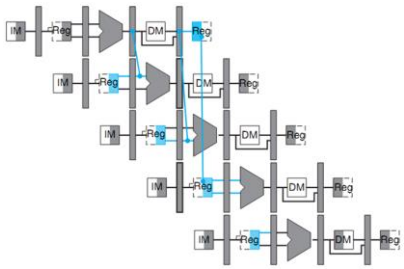
注意：
多周期和流水线指令周期都不定长。
一个周期完成多条指令？

Pipelining Hazards：类型，原因，处理

- “A *hazard*（相关，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

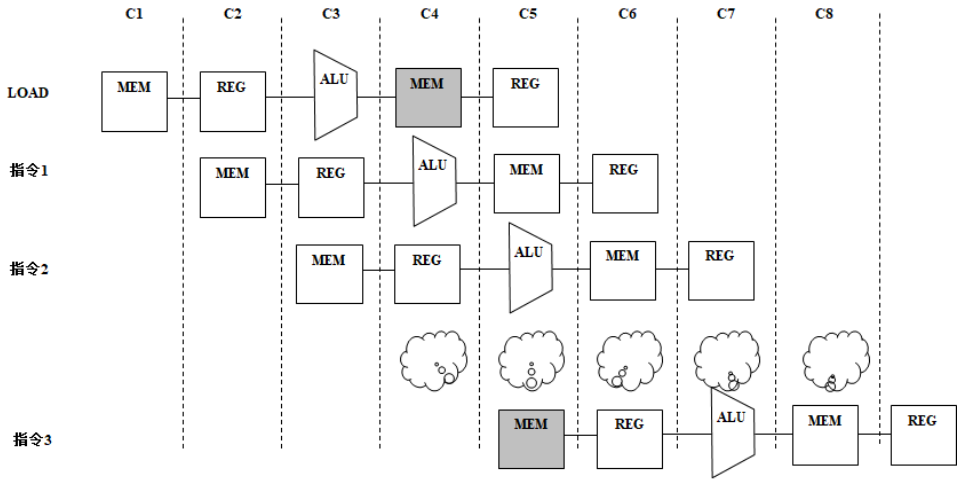


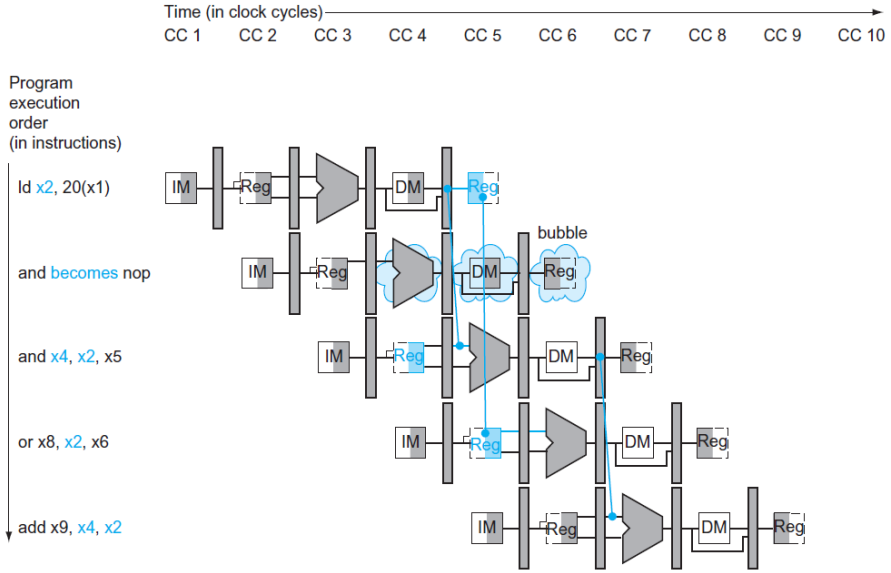
  - *Data hazard*，数据依赖
    - Data dependency between instructions ：生产者消费者
      - Need to wait for previous instruction to complete its data write
    - Forwarding: grab operand from Interstage-Buf, rather than RegFile.
      - Multilevel Bypass: EX bypassing，Load/Store Bypassing，beq bypassing
      - 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

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？

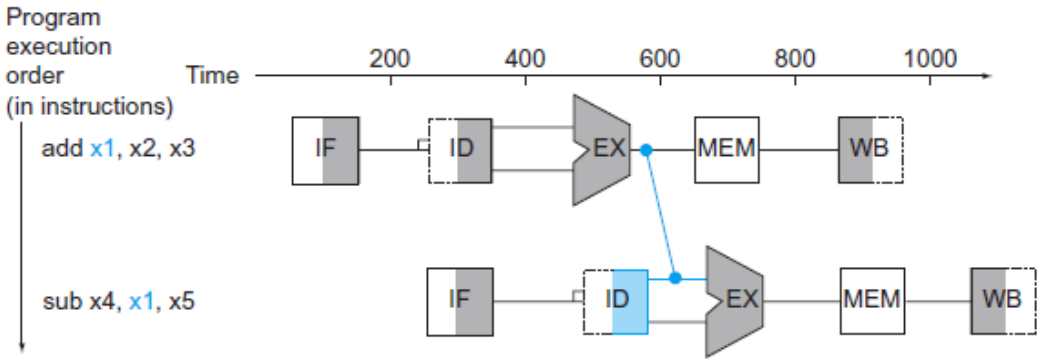
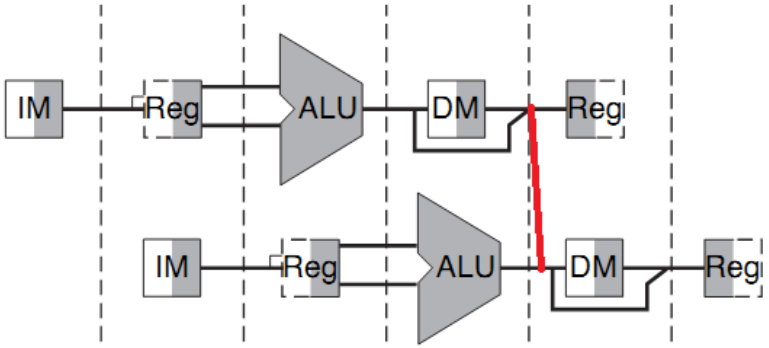


图4-27：EX/MEM forwarding to EX

数据通路

前推到哪儿，从哪儿前推？

FW控制器



lw-sw：MEM/WB forwarding to MEM

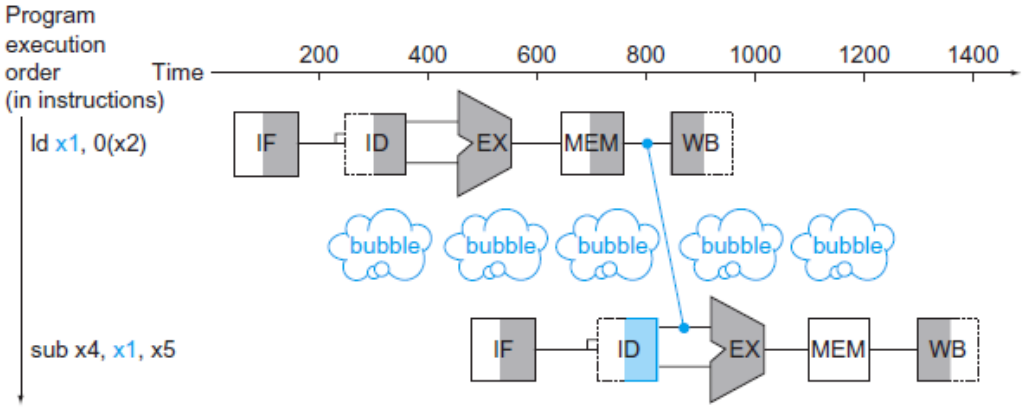


图4-28：stall，MEM/WB forwarding to EX

Benchmark: SPEC2006

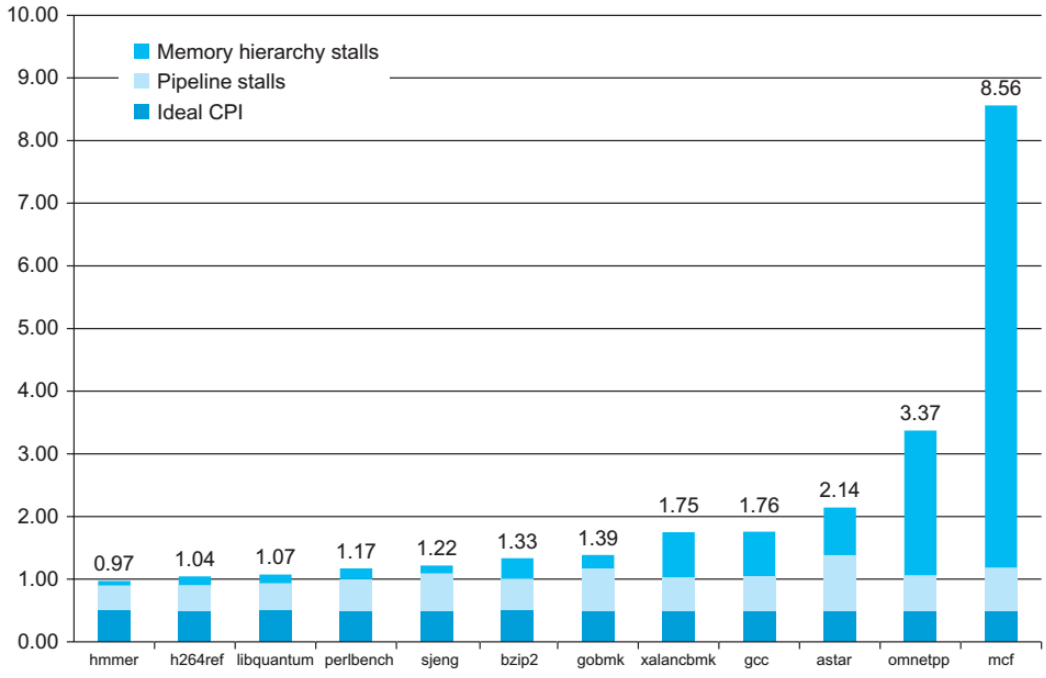


图4-73 CPI of ARM Cotex-A53
CPI：理想0.5，平均1.3
停顿：60%冒险，40%访存

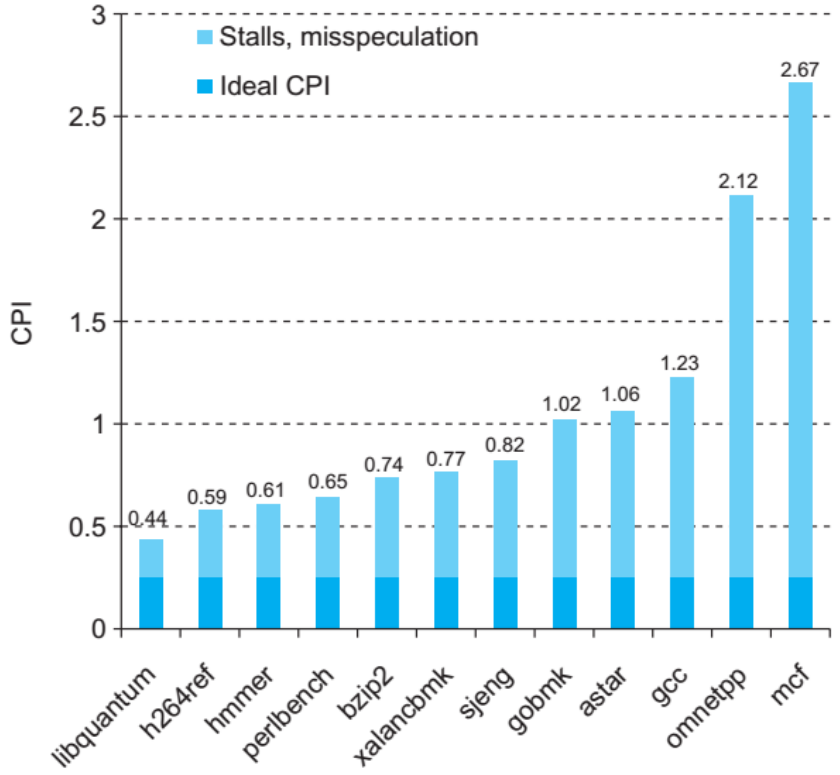
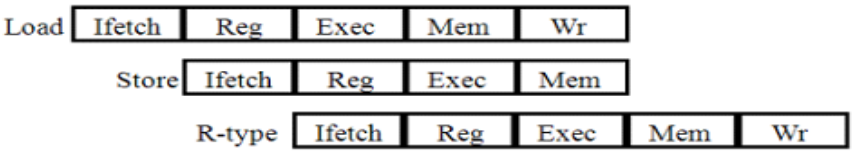
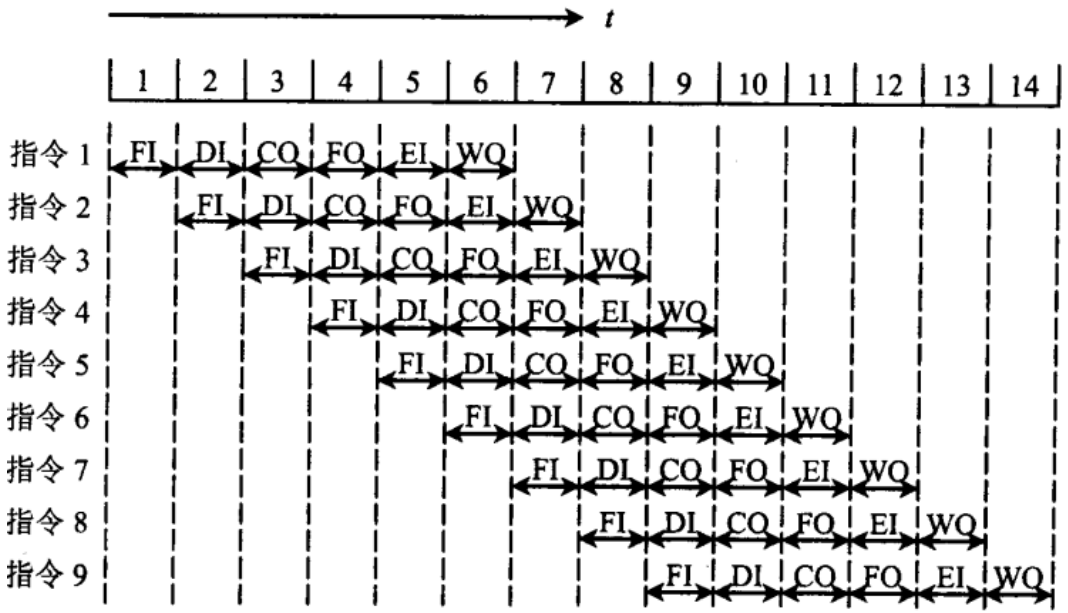


图4-75 CPI of Intel Core i7，
CPI：理想0.25，平均0.79

流水线中的多发技术，$4.10

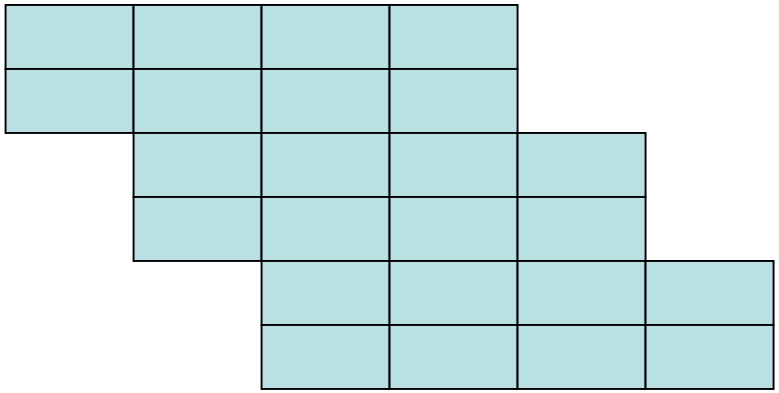
- 目的：“在一个时钟周期（机器周期？）内流出多条指令”
  - 理想流水线 IPC=1，能否使 IPC > 1
- 常见的多发技术(multiple issue)
  - 超标量技术：动态多发射
  - 超流水线技术
  - 超长指令字技术 ：静态多发射





超标量技术（ superscalar ）

- 简单标量处理器：在任一时刻取指并执行一条指令的简单的处理器
- 超标量：指在每个时钟周期内可同时发射并执行多条指令
  - 系统结构要求：处理机中配置多个功能部件和指令译码电路，以及多个寄存器端口和总线，以便能实现同时执行多个操作
  - 编译器要求：决定哪几条相邻指令可并行执行。



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

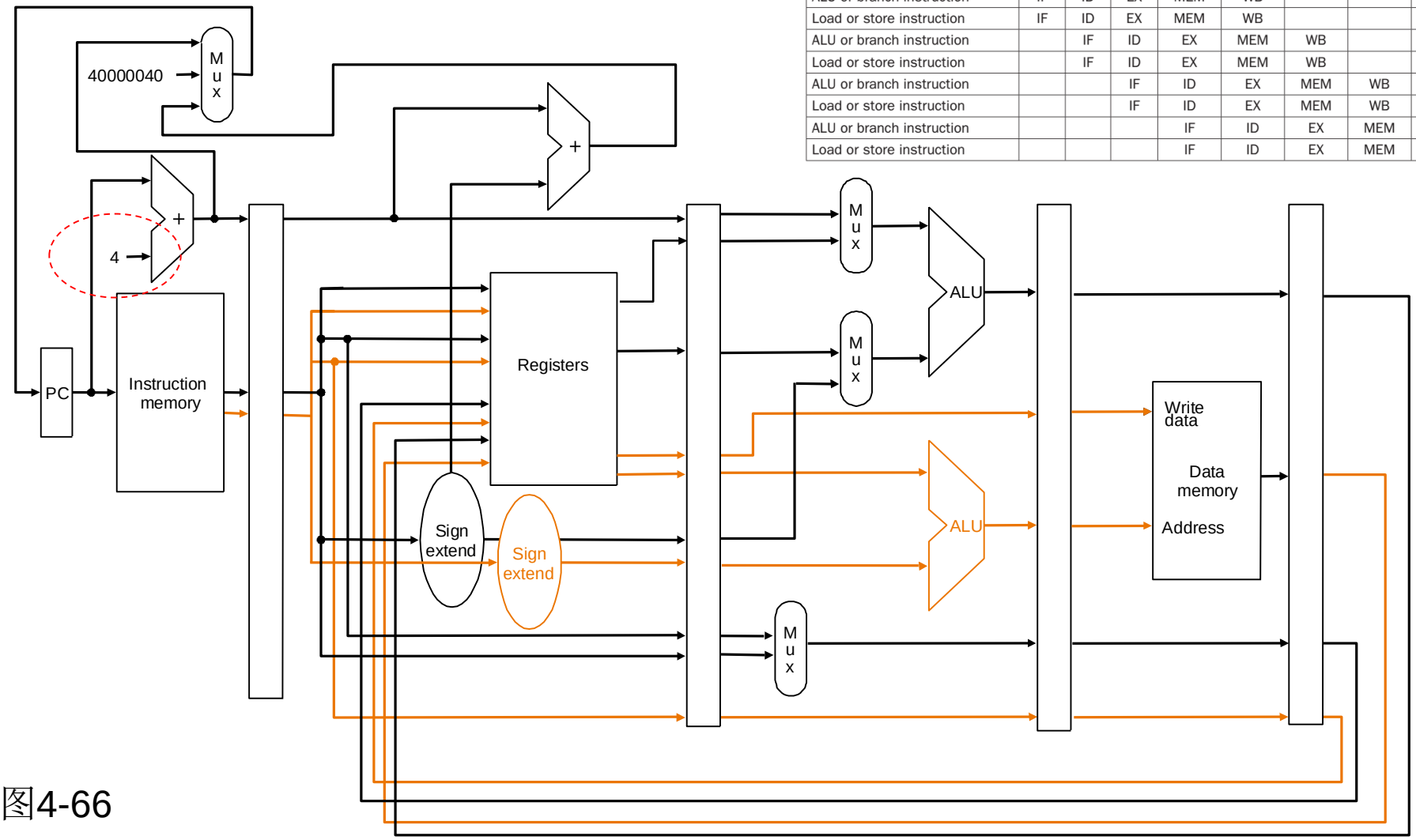
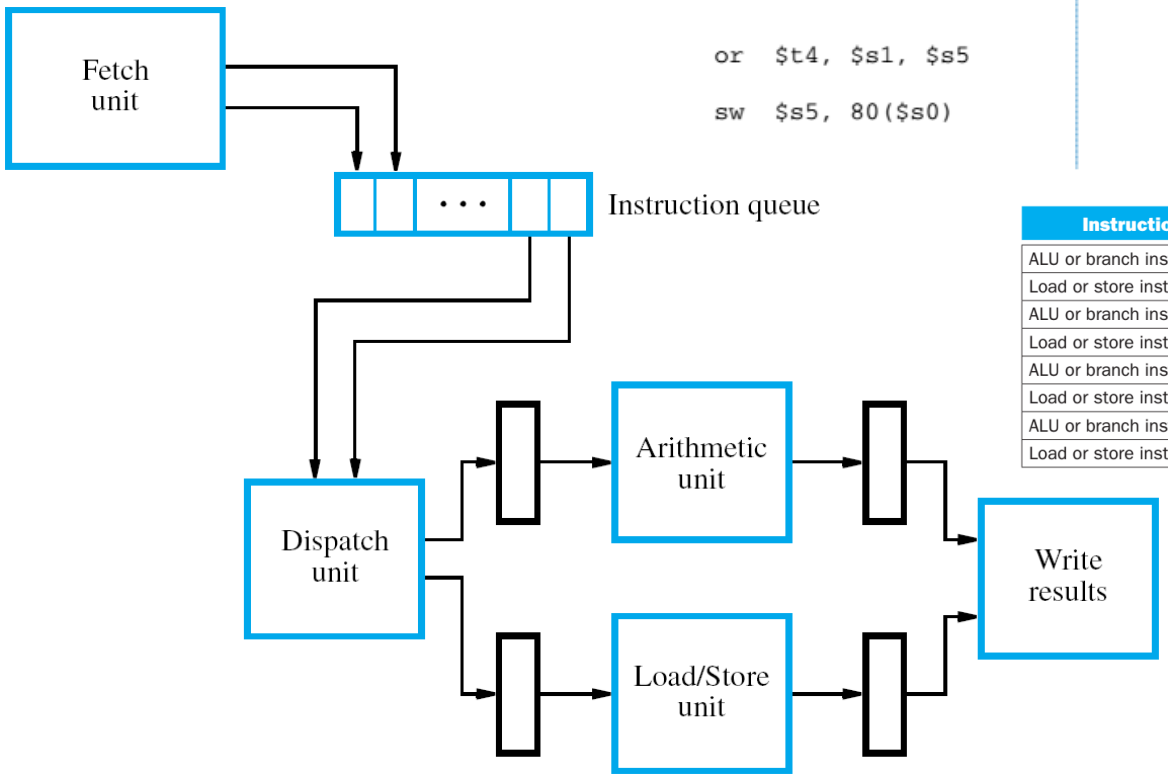
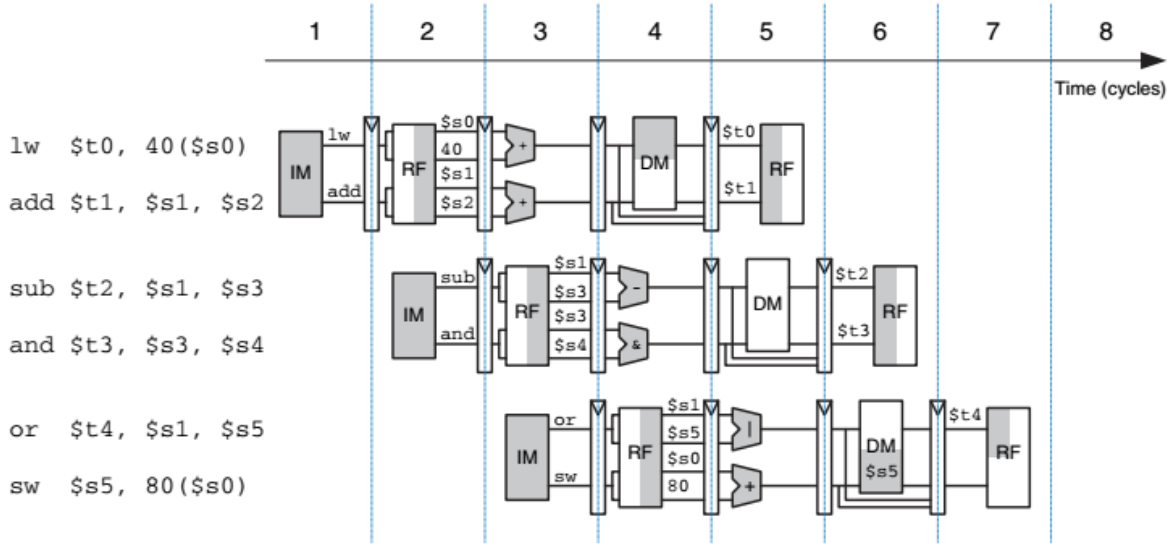


图4-66

A Two-Way Superscalar Pipeline

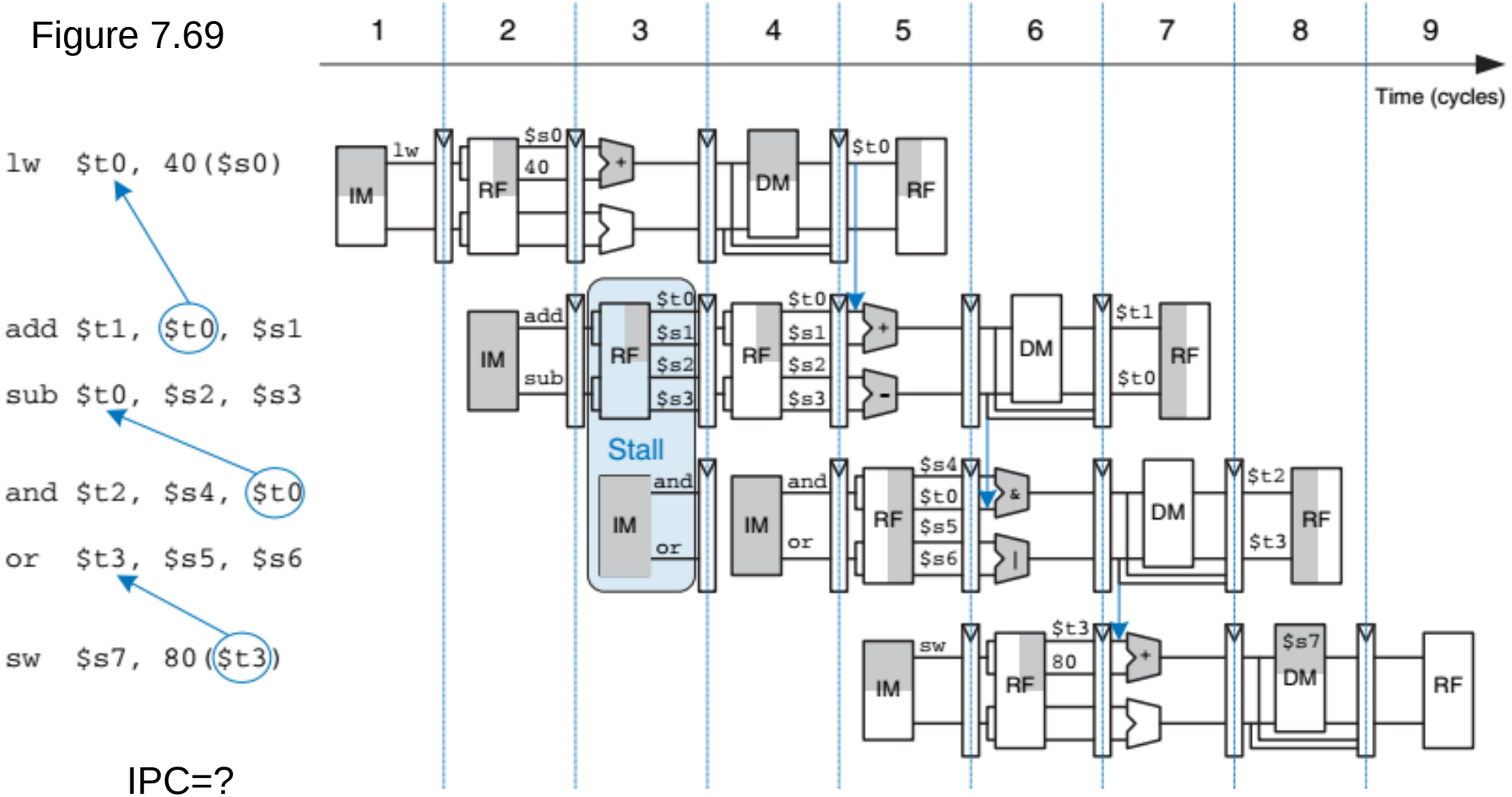




Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM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

Program with data dependencies

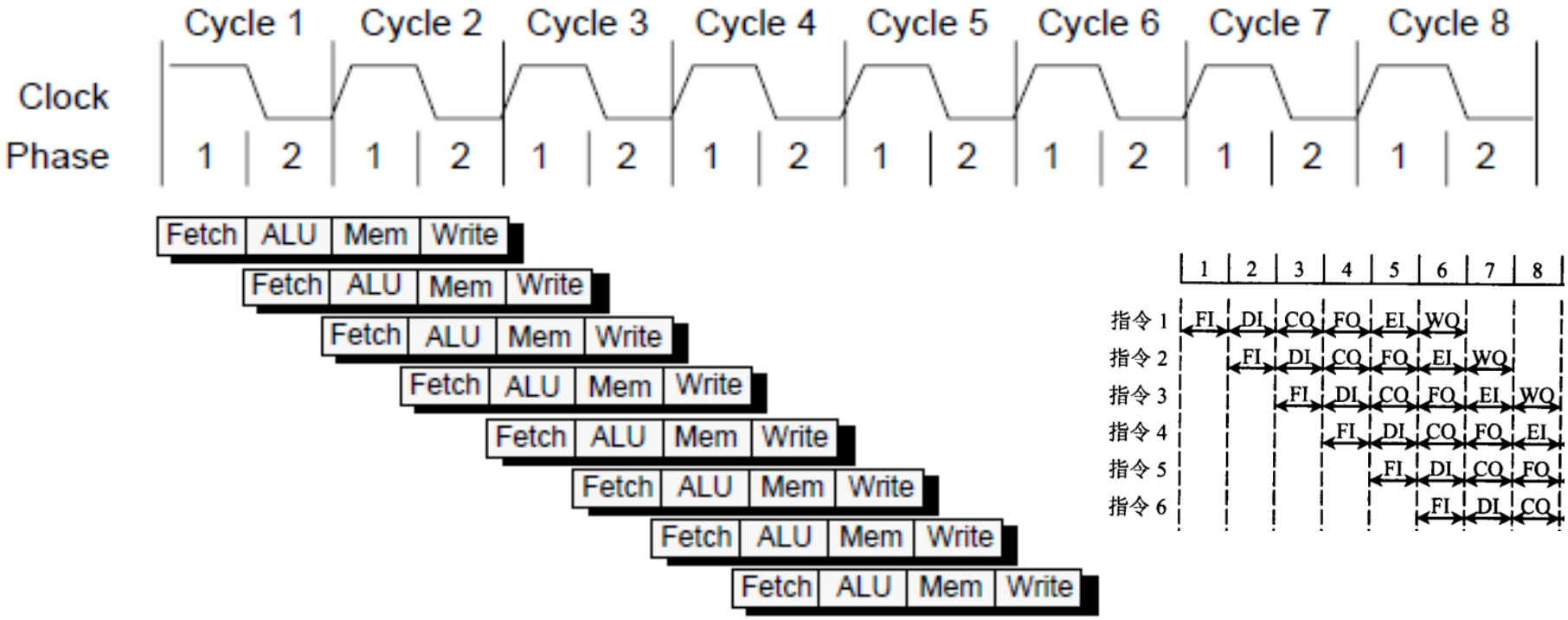
Figure 7.69



IPC=?

David Harris, et al. Digital Design and Computer Architecture, 2ed, Elsevier, 2013

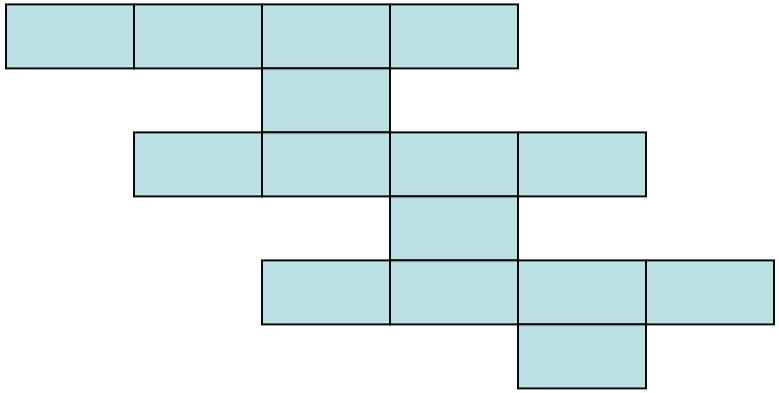
Four-Deep Superpipeline



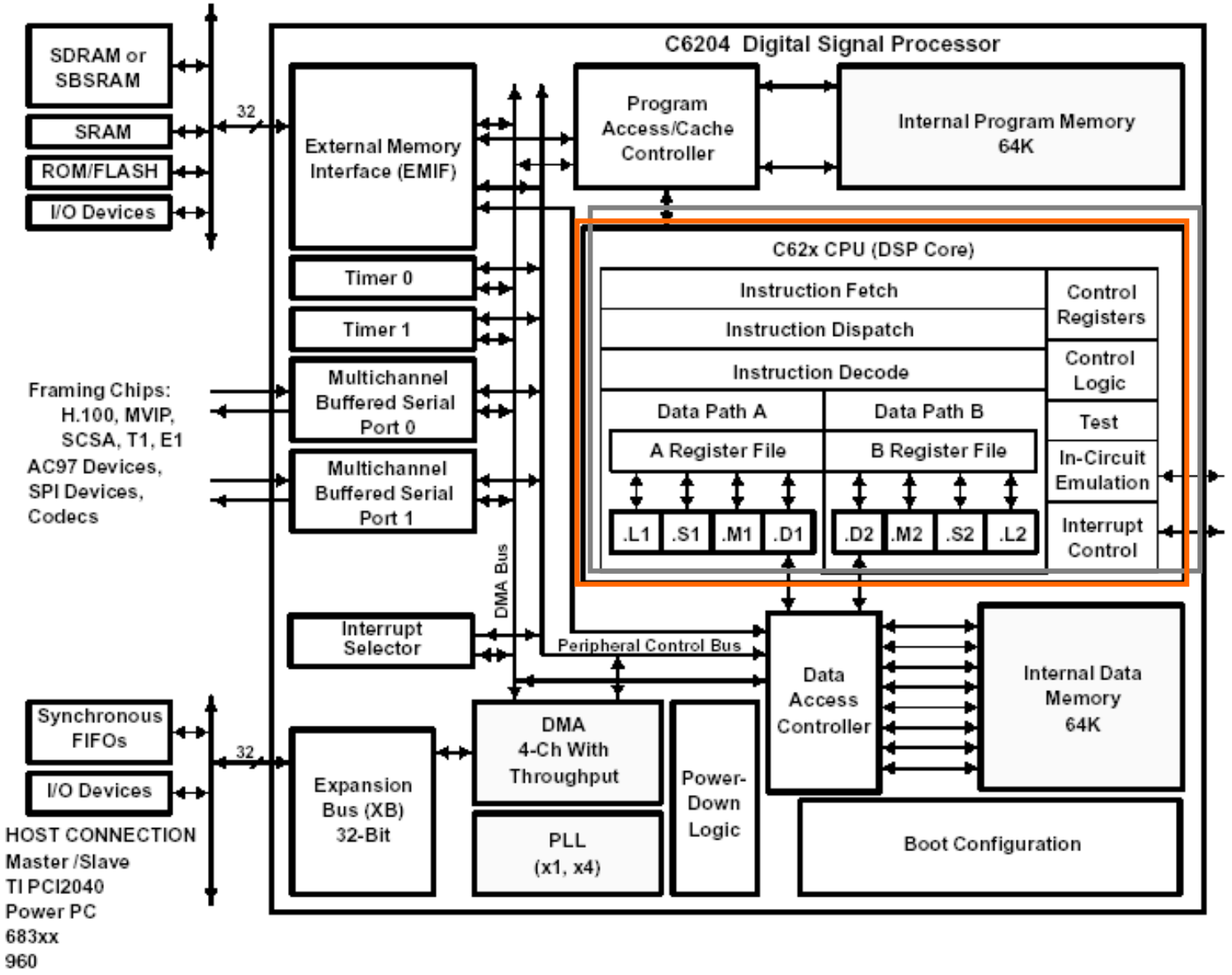
- 超流水线

超长指令字技术

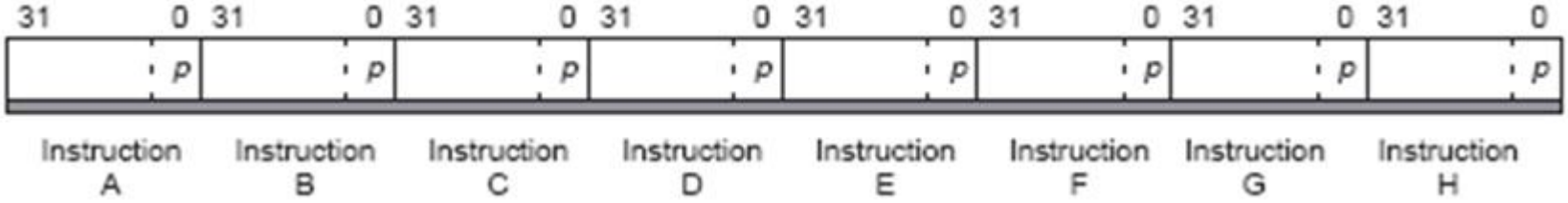
- **VLIW**把多条能并行操作的指令组合成一条具有多个操作码字段的超长指令**(**指令字长可达几百位**)**，由这条超长指令控制**VLIW**机中多个独立工作的功能部件，由每一个操作码字段控制一个功能部件，相当于同时执行多条指令。
- 超长指令字**(VLIW)**技术和超标量技术都是采用多条指令在多个处理部件中并行处理的体系结构，在一个时钟周期内能流出多条指令。



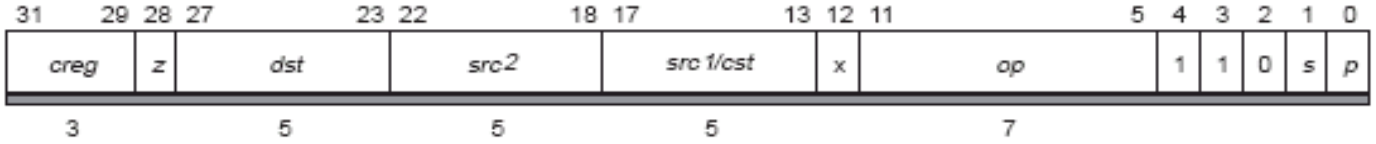
TMS320C64x：VLIW



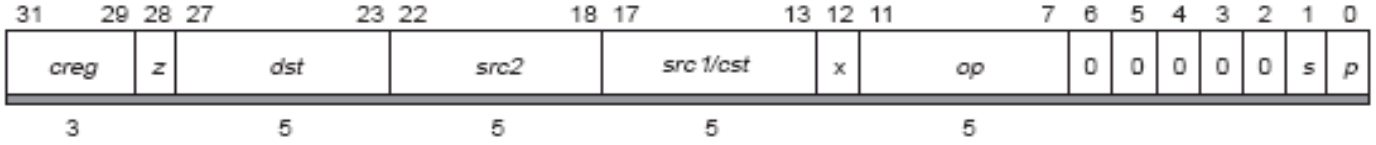
TMS320C64x指令字



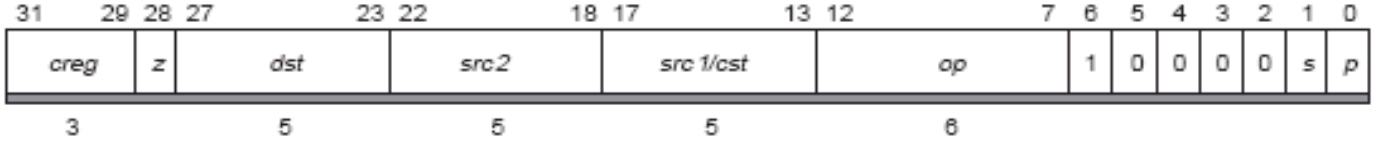
Operations on the .L unit



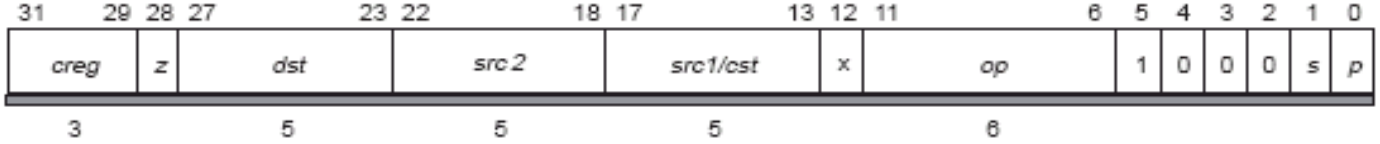
Operations on the .M unit



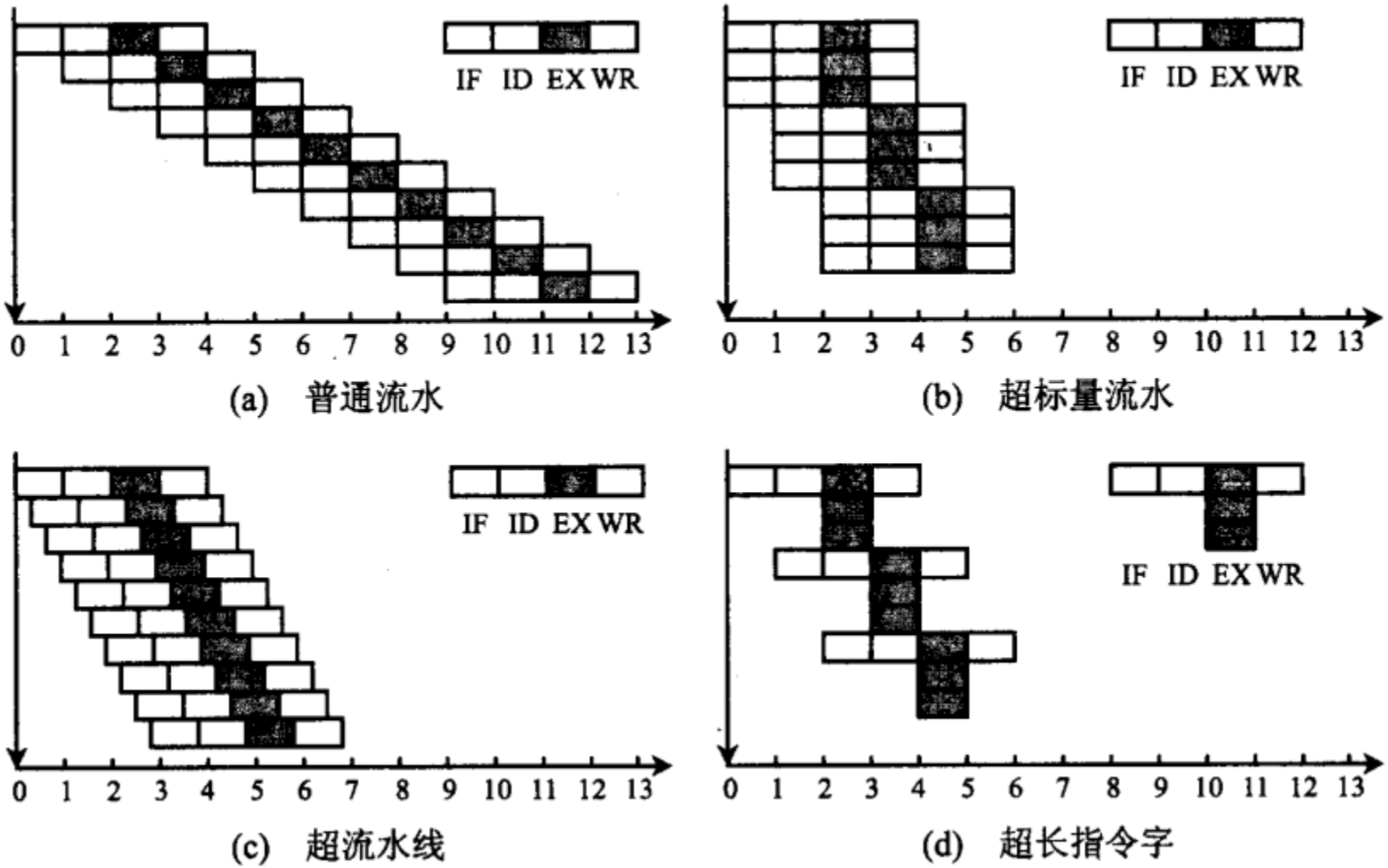
Operations on the .D unit



Operations on the .S unit



四种流水技术比较：图8.20@唐

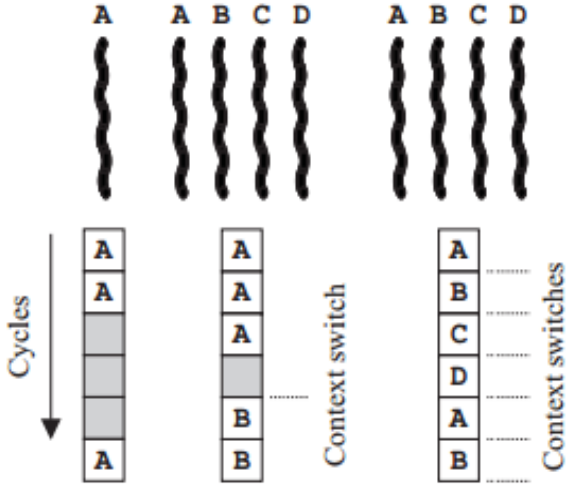


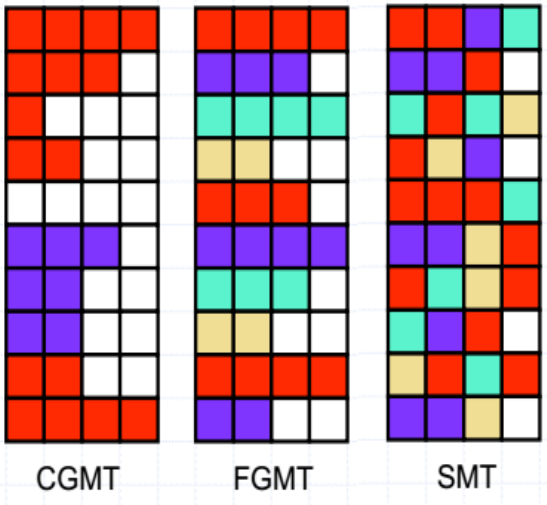
执行时间？

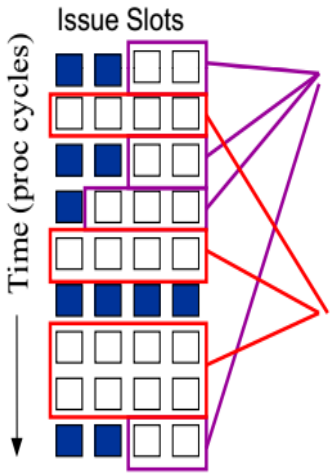
- add x2，x1，x1
- add x3，x2，x2
- ld x4，x3，0；*ld x4，0(x3)*
- add x5，x4，x4

- 无冒险处理部件
- 有冒险处理部件（FW，interlock）

HMT技术（RV $6.4）：MIMD

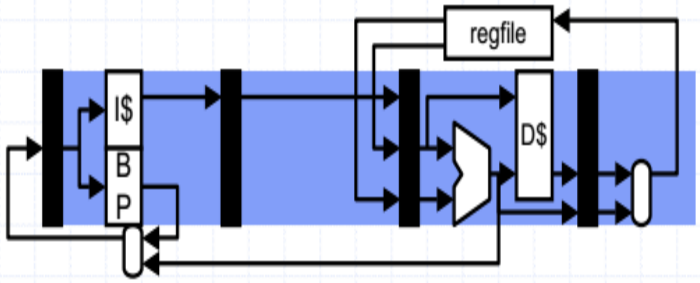


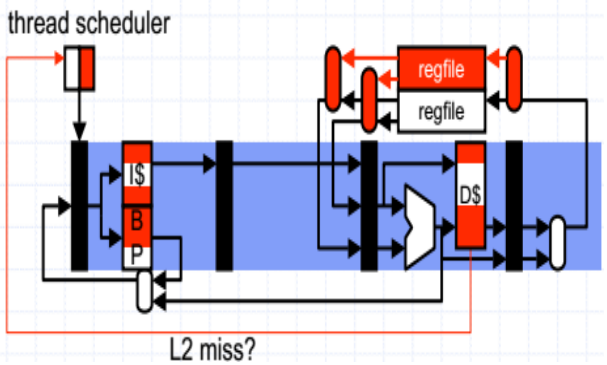


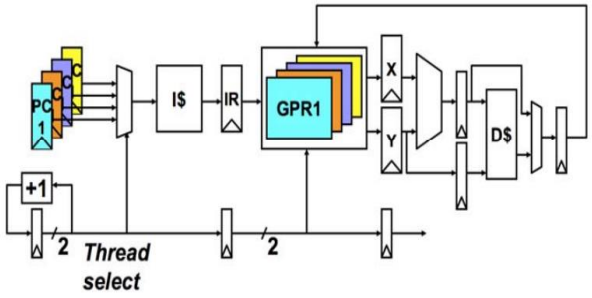


4个硬件线程
类似图6-5
H&P图3.8

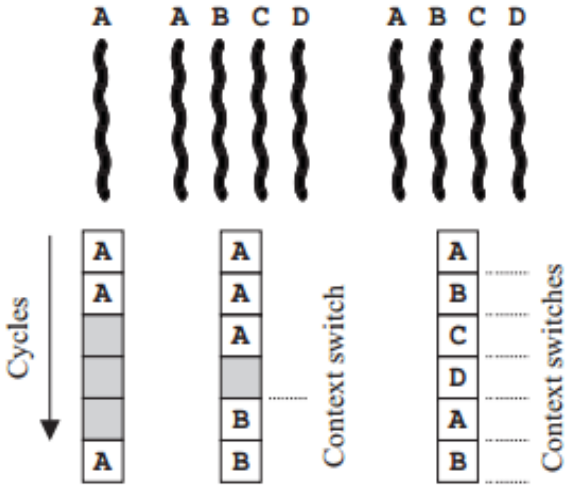
- **coarse-grain multithreading，Block Multithreading**







FGMT消除数据依赖和分支依赖

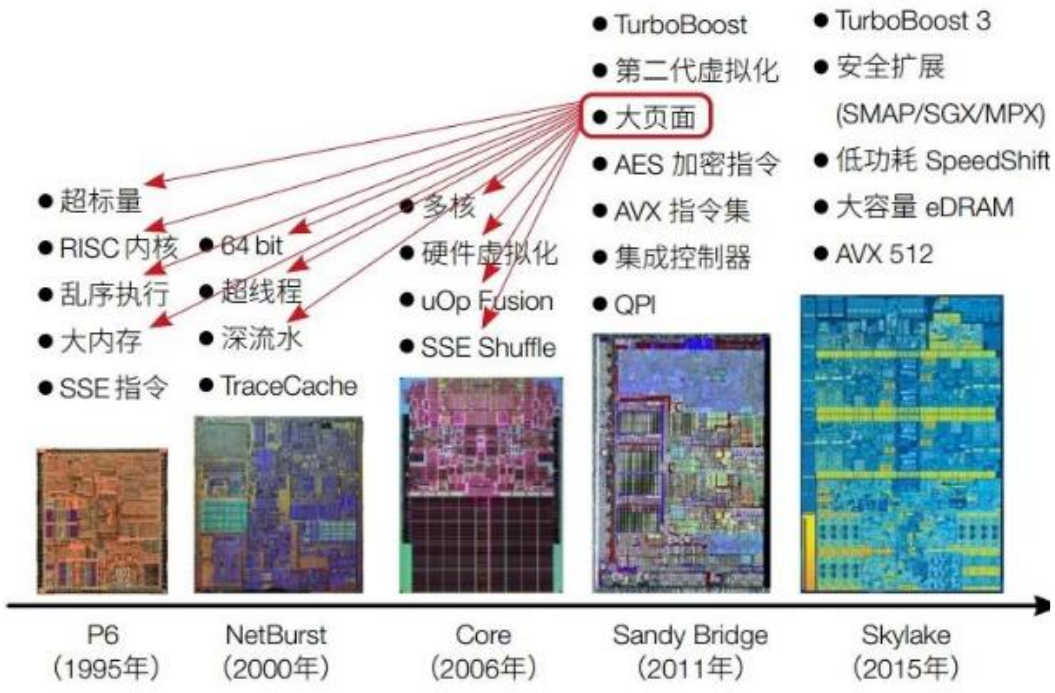


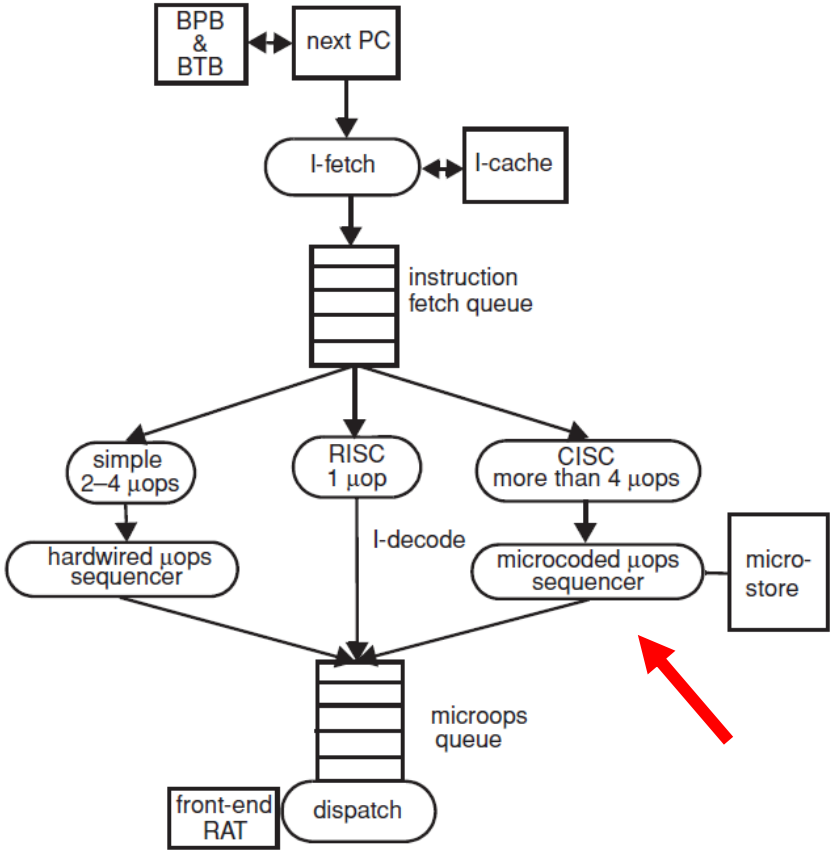
TID	Addr.	Inst.	Cycle							
			1	2	3	4	5	6	7	8
0	0x00	BR 0x0C	F	D	E	M	W			
0	0x04	I		F	D	-	-	-		
0	0x08	I			F	-	-	-	-	
0	0x0C	I				F	D	E	M	W

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

- 提升了总吞吐量，但也增加了单线程延迟

Intel处理器架构演化（1995—2015 年）





- 复杂ISA的前端

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