Chap.4 The Processor

4.7 Data Hazards: Forwarding versus Stalling

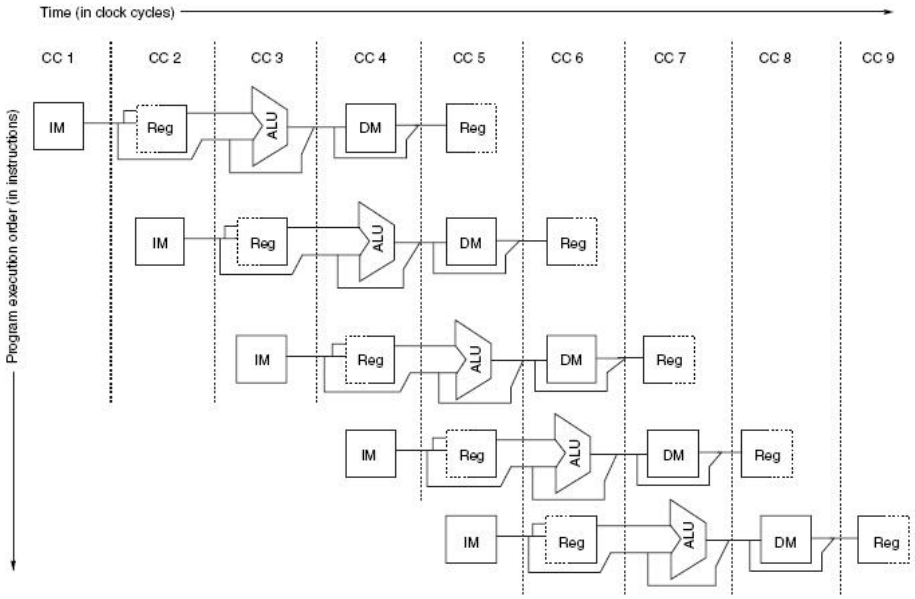
- 在流水线的实现中有哪些困难？
  能够如何解决？

- 本次课程主要内容：
  流水线冒险（Hazards）及其解决策略

上次课流水线实现的基本原理回顾

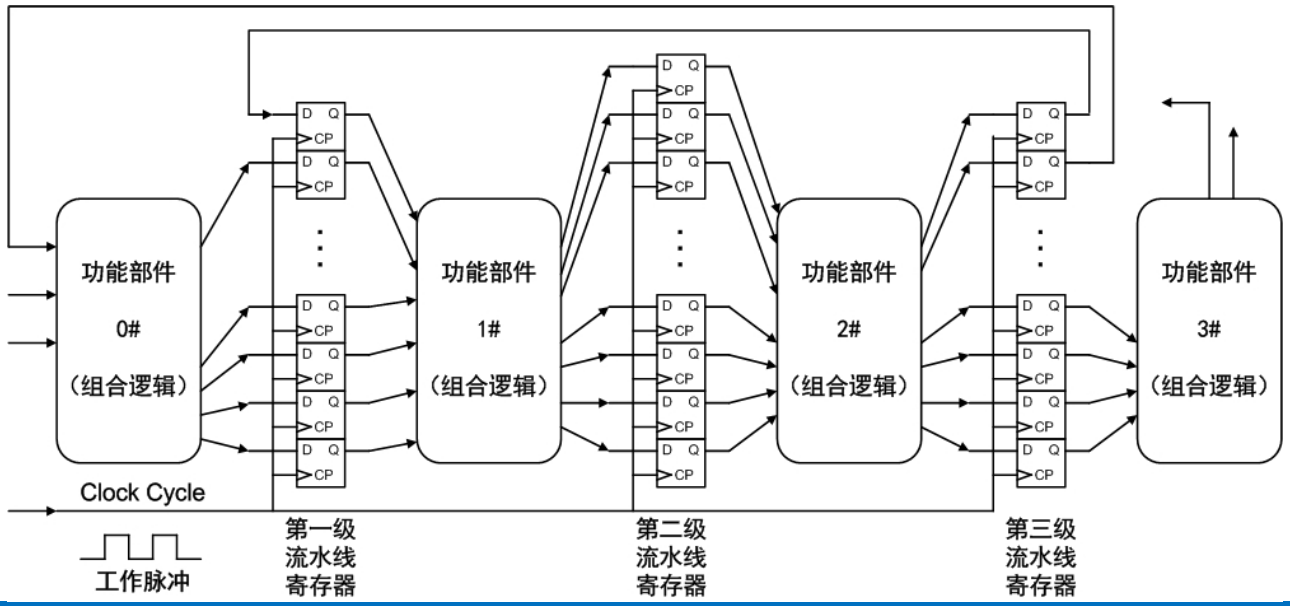
实现指令流水的两个最基本问题

1. 分段 — 为了重叠起来并行执行　　（划分工位）
2. 流动 — 能够向前同步推进执行/流水　（段间接口）

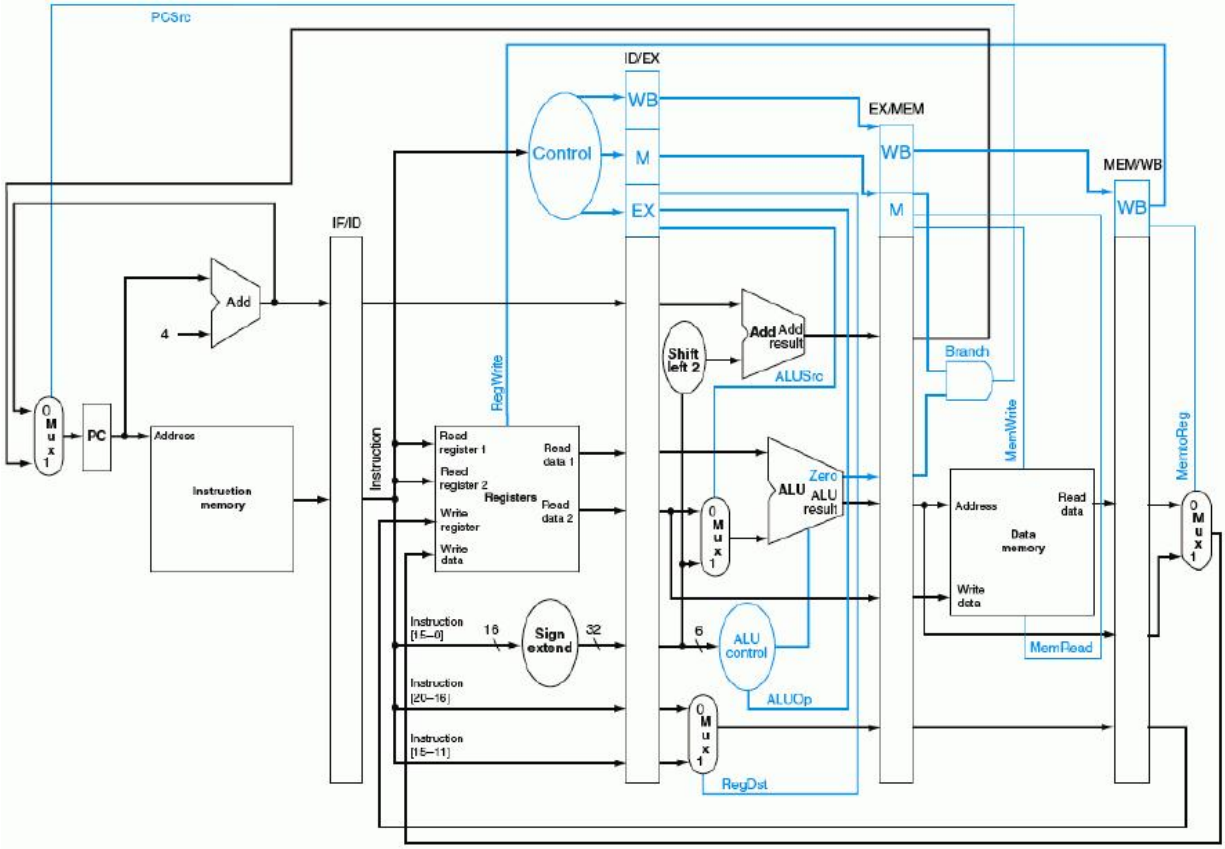


流水线寄存器的电路结构原理示意简图

流水线寄存器在各流水段间起分割墙的作用。每段的所有执行结果都保存在流水线寄存器中，在下一个时钟周期作为下一个流水段的输入。通过流水线寄存器，后级流水段也可以向前级反馈数据和控制信息。



含有控制信号的流水线数据通路原理简图



理想的指令流水线执行状况

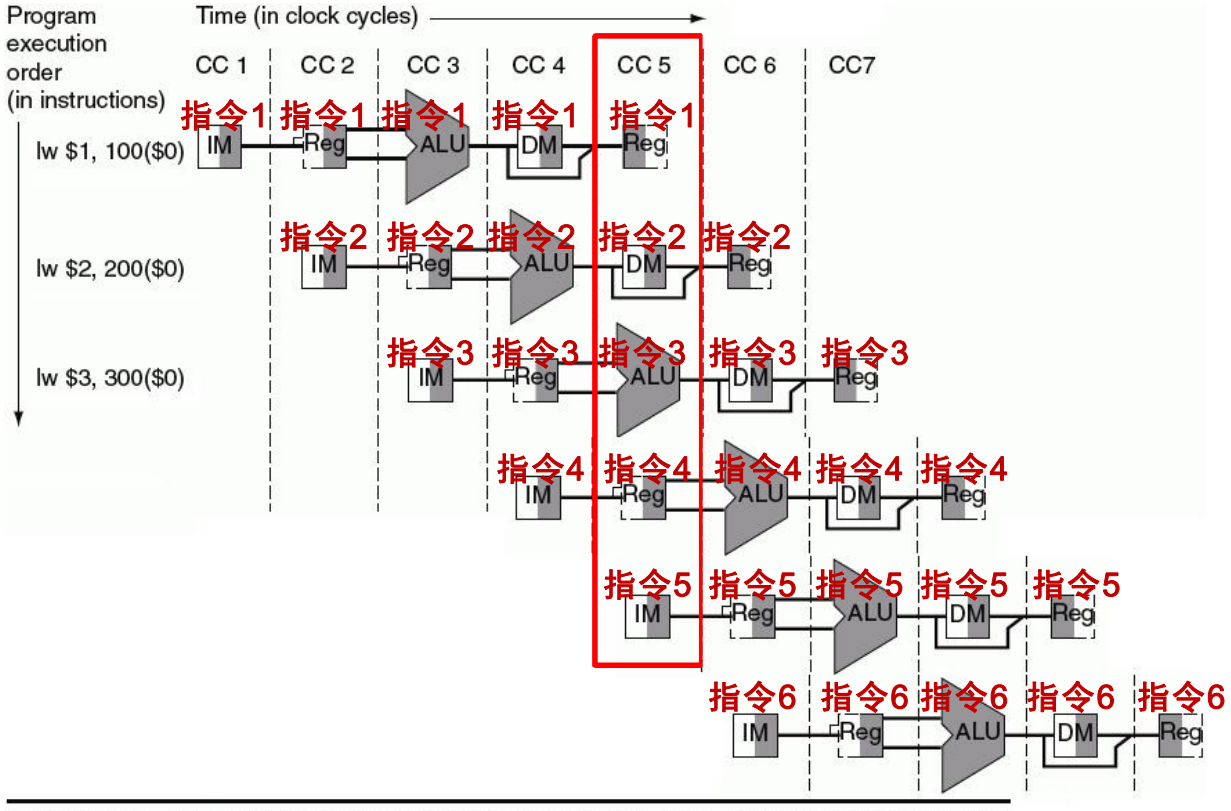


FIGURE 6.10 **Instructions being executed using the single-cycle datapath in Figure 6.9, assuming pipelined execution.**

本次课程主要内容

在流水线的实现中有哪些困难？能够如何解决？

流水线冒险（Hazards）及其解决策略

流水线冒险（Hazards）

1. 结构冒险：也叫结构冲突，当硬件在指令重叠执行中不能支持指令所有可能的组合时，所发生的资源冒险。
2. 数据冒险：在同时执行的几条指令中，一条指令依赖于前一条指令的结果数据，却得不到时，发生的冒险。
3. 控制冒险：流水线中的转移指令或其它改写PC的指令所造成的冒险。

- 流水线中的“冒险”，可能会引起流水线的“停顿”，从而影响流水线的性能。

流水线冒险（Hazards）

结构冒险（structural hazard）

- An occurrence in which a planned instruction cannot execute in the proper clock cycle because the hardware cannot support the combination of instructions that are set to execute in the given clock cycle.
- Arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution.

流水线冒险（Hazards）

结构冒险（structural hazard）

当处理器进行处理流水时，指令的重叠执行要求功能单元能够流水，而且资源重复设置，以便流水线中的指令能自由组合。如果因为资源冲突而无法使用某种指令的组合，那么该处理器就被称为是有结构冒险的。

例如，一台机器只有一个寄存器堆写入端口，但在某些情况下，如果流水线可能要求在一个时钟周期写入两次，这将会产生一个结构冒险。

结构冒险（structural hazard）

【示例】当某流水线机器的指令和数据共享同一个存储器，当一条指令包含有数据访问时，它将可能与下一条指令的取指令冲突。

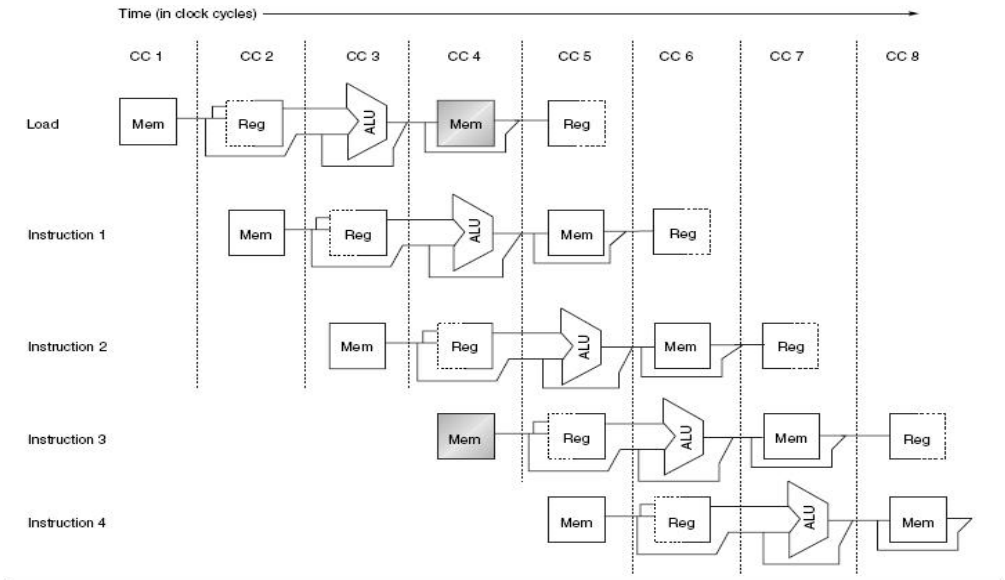


Figure A.4 A processor with only one memory port will generate a conflict whenever a memory reference occurs. In this example the load instruction uses the memory for a data access at the same time instruction 3 wants to fetch an instruction from memory.

结构冒险（structural hazard）

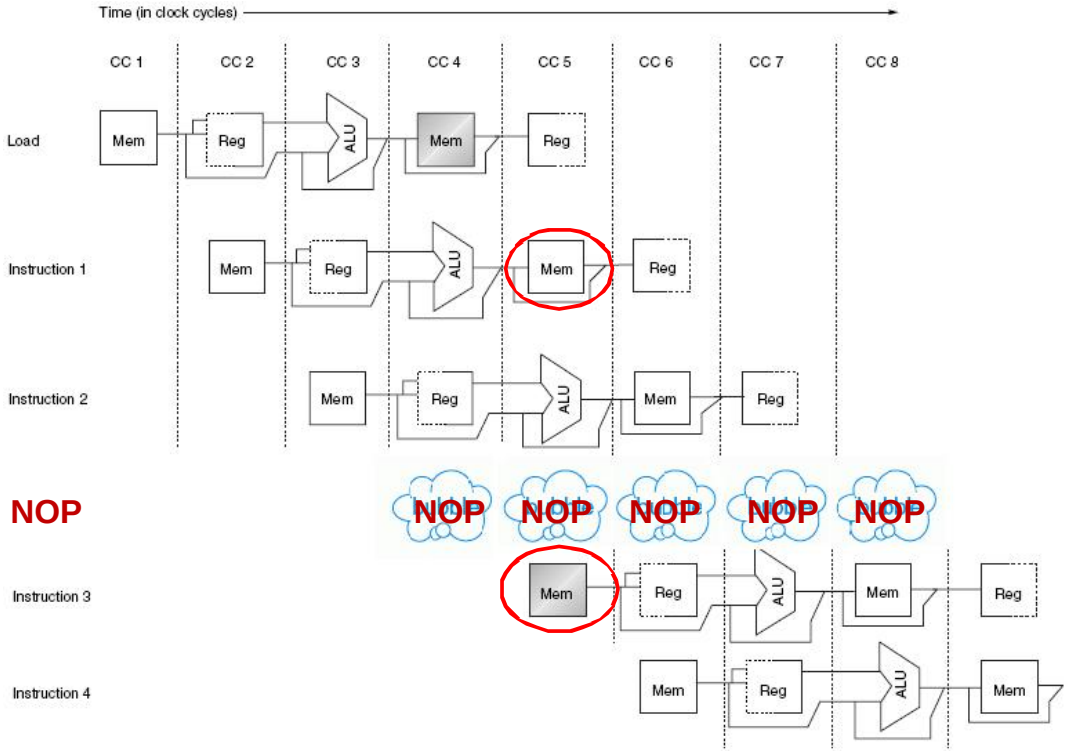
一旦发生了结构/资源冒险，流水线将停顿其中的一条指令，直到所需的功能单元能够使用为止。这种停顿将把CPI从理想值1增大。

在上页的存储器资源冲突中，为消除这种结构冒险，当需要访问存储器时就把流水线停顿一个时钟周期。

停顿(*stall*)通常称为流水气泡(*pipeline bubble*)或气泡(bubble)，因为它在流过流水线的过程中只是占据了空间而不做实际有效的工作。（类似于一条NOP指令）。

结构冒险（structural hazard）

通过插入一个停顿周期来解决冲突



结构冒险（structural hazard）

表示停顿的一种简化流水线图

	Clock cycle number									
Instruction	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i+1$		IF	ID	EX	MEM	WB				
Instruction $i+2$			IF	ID	EX	MEM	WB			
Instruction $i+3$				stall	IF	ID	EX	MEM	WB	
Instruction $i+4$						IF	ID	EX	MEM	WB
Instruction $i+5$							IF	ID	EX	MEM
Instruction $i+6$								IF	ID	EX

Figure A.5 A pipeline stalled for a structural hazard—a load with one memory port.

结构冒险（structural hazard）

load结构冒险对流水线所造成的影响

【例题】：假设数据引用占指令的40%，在忽略结构冒险的前提下流水线的理想CPI是1。若有结构冒险的处理器的时钟频率是无结构冒险的处理器的时钟频率的1.05倍，假设没有别的性能损耗，那么两种流水线哪个更快？快多少？

结构冒险（structural hazard）

load结构冒险对流水线所造成的影响

【例题】：假设数据引用占指令的40%，在忽略结构冒险的前提下流水线的理想CPI是1。若有结构冒险的处理器的时钟频率是无结构冒险的处理器的时钟频率的1.05倍，假设没有别的性能损耗，那么两种流水线哪个更快？快多少？

【解答】有结构冒险的处理器的指令平均执行时间为：

$$\text{指令平均执行时间} = \text{CPI} \times \text{时钟周期长度}$$

$$= \left(1 + 0.4 \times 1\right) \times \frac{\text{理想时钟周期长度}}{1.05}$$

$$= 1.33 \times \text{理想时钟周期长度}$$

显然，没有结构冒险的处理器更快。

【注】对该结构冒险的一种改进方案是为指令提供单独的存储器。具体是把Cache分为指令cache与数据cache，或者用一个单独的缓冲栈来保存指令，称为指令缓冲栈。（后续伏笔）

流水线冒险（Pipeline Hazards）

数据冒险（data hazard）

- An occurrence in which a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

- Arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

流水线冒险（Pipeline Hazards）

数据冒险（data hazard）

- 流水线通过指令的重叠执行，就有可能改变指令的相对执行时间或某些环节的相对执行顺序。就有可能导致后面被提前执行的指令，出现了不能及时得到前面指令执行结果的情况，从而导致流水线不得不停顿等待的情况。这种冲突称为数据冒险。
- 【示例】考察如下代码的执行：

```
DADD   R1,R2,R3      ; [R1] ← [R2] + [R3]
DSUB   R4,R1,R5      ; [R4] ← [R1] - [R5]
AND    R6,R1,R7      ; [R6] ← [R1] & [R7]
OR     R8,R1,R9      ; [R8] ← [R1] | [R9]
XOR    R10,R1,R11    ; [R10] ← [R1] ^[R11]
```

数据冒险示例

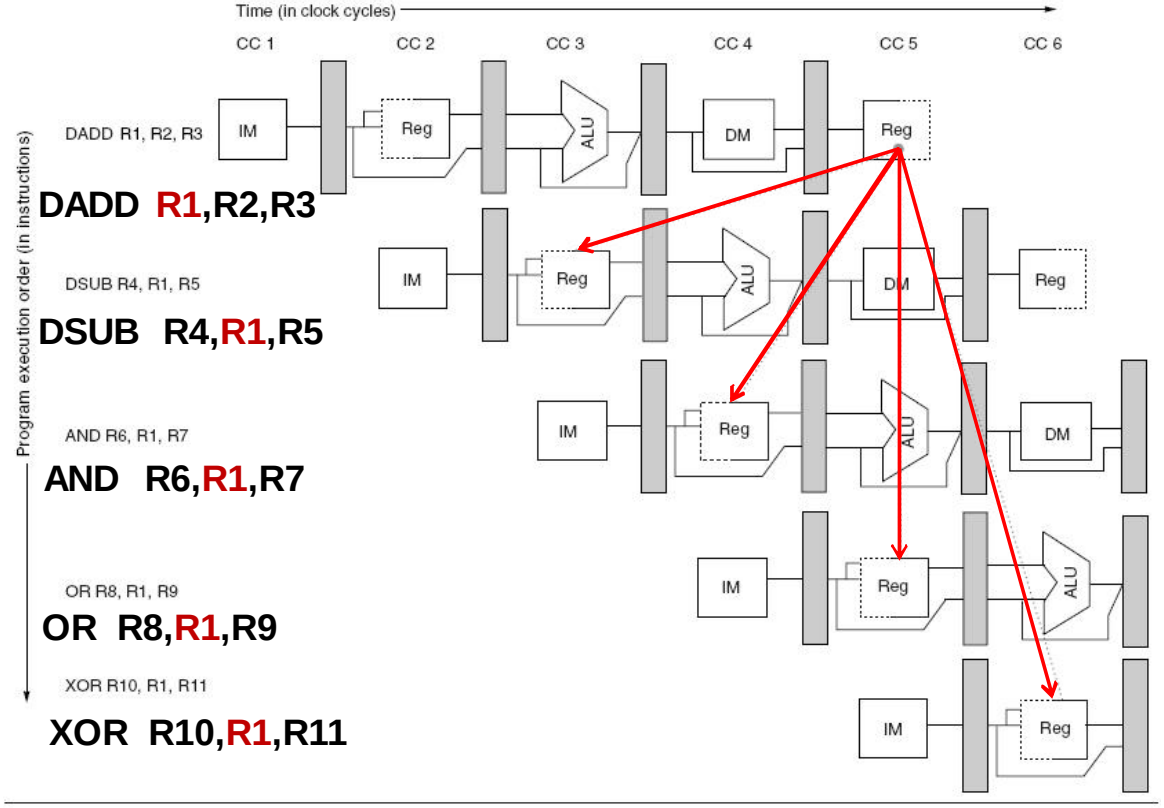


Figure A.6 The use of the result of the DADD instruction in the next three instructions causes a hazard, since the register is not written until after those instructions read it.

数据冒险

数据冒险的消除策略

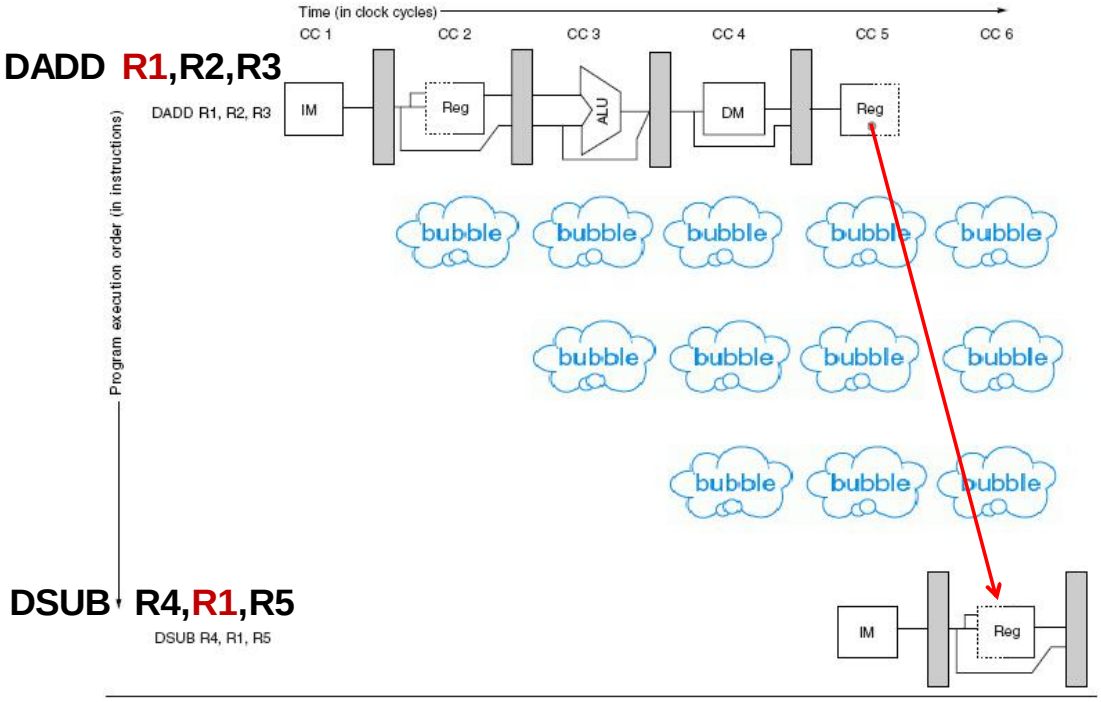
- 最简单的方法：

加入气泡

【缺点】：引入了延迟

消除数据冒险的策略

加入气泡——引入了3个气泡的时间延迟

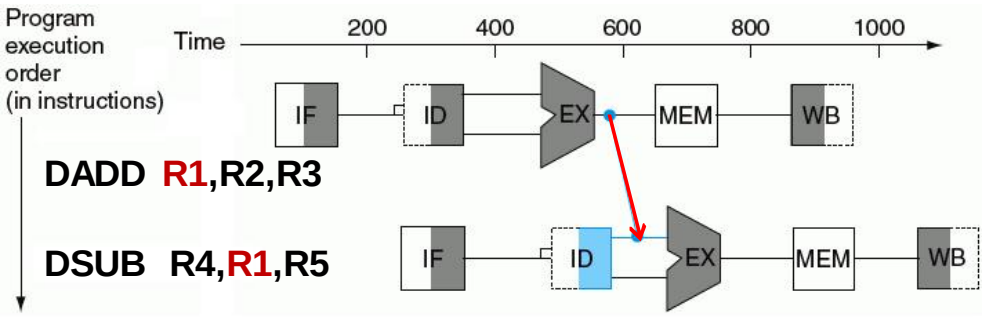


消除数据冒险的策略

进一步分析前面示例的数据引用关系，发现：

对DADD的运算结果需求最紧要的DSUB指令，其在EX阶段所需要的R1中的数据，在上一个时钟周期的ALU操作结束时，其实已经生成了。

如果能够：



消除数据冒险的策略

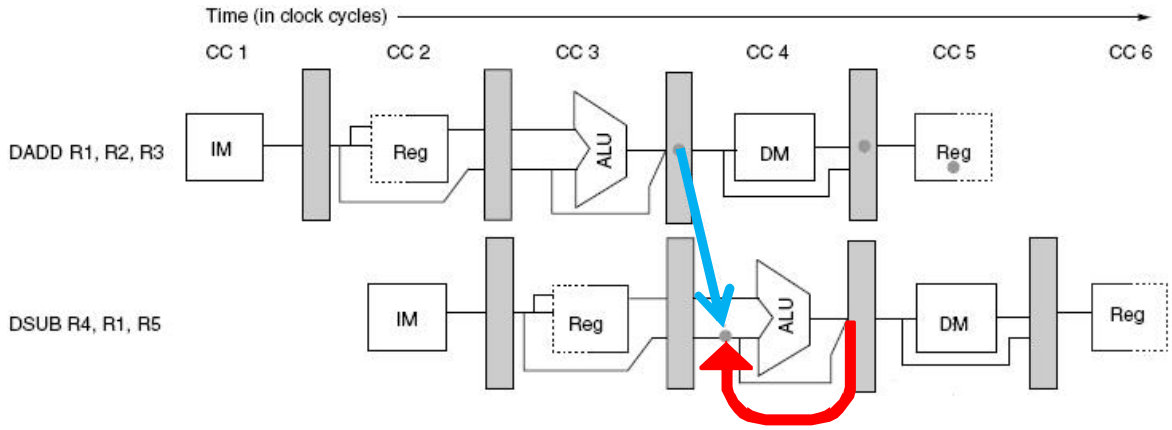
使用直通技术解决数据冒险引起的停顿

上例中的直传企图可以用简单的硬件技术来解决，这种技术称为直通（*forwarding*），或称为旁路（*bypassing*）或短路（*short-circuiting*）。

其技术关键是注意到了DSUB操作是在DADD操作产生了结果以后才真正使用这个结果的。

如果把DADD的结果从EX/MEM寄存器直接传送到DSUB需要的地方，即ALU的输入锁存器，那么就没有必要引入停顿了。

使用直通技术解决数据冒险引起的停顿



1. 送入到EX/MEM流水线寄存器的ALU结果，总是反馈到ALU的输入选择器端。
2. 如果直通硬件检测到前一次的ALU操作写入的寄存器正好是当前ALU操作的源操作数，那么控制逻辑就选择直通结果作为ALU的输入，而不用去从寄存器堆中读取源操作数。

使用直通技术解决数据冒险引起的停顿

继续分析前面示例的数据引用关系，发现：

需要直通的操作数可能不只来自最近的操作，而且可能来自于前面3个周期前启动的操作。

可以使用直通技术解决的数据冒险

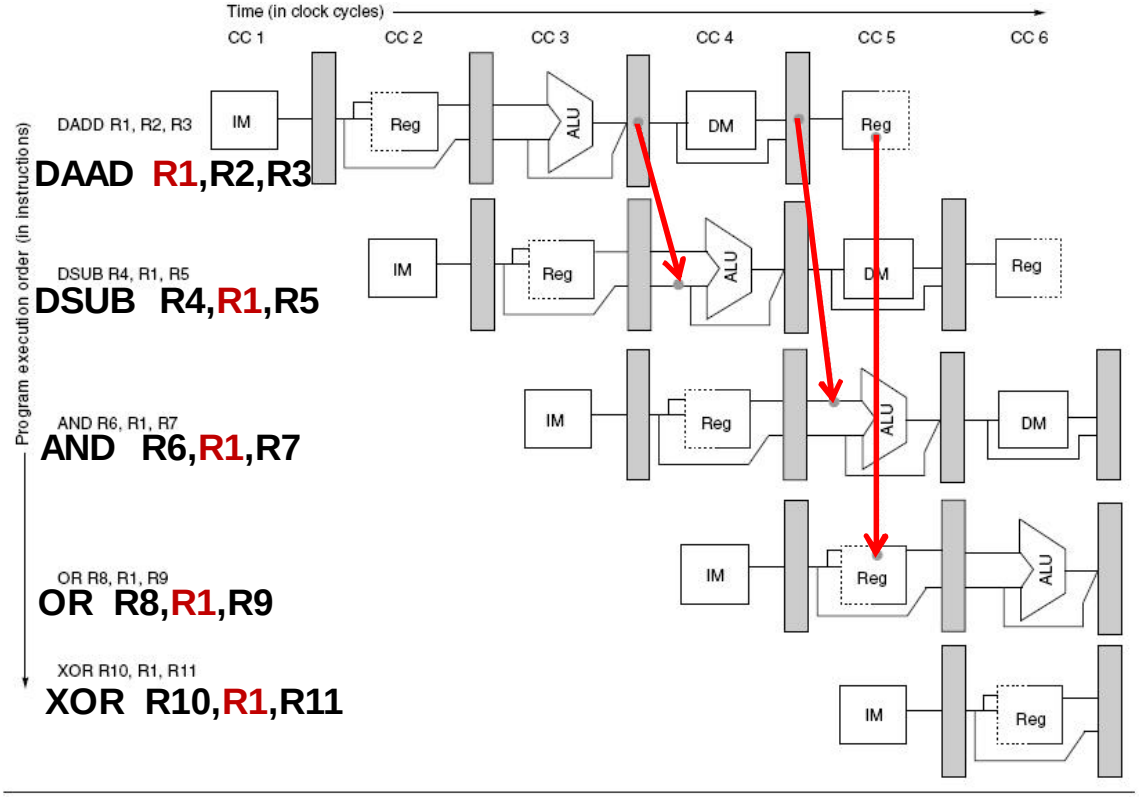


Figure A.7 A set of instructions that depends on the DADD result uses forwarding paths to avoid the data hazard.

使用直通技术解决数据冒险引起的停顿

进一步扩展和一般化直通思想，可以把结果直接送到需要它的功能单元：

一个结果能够从一个单元输出所对应的流水线寄存器直接送到另一个单元的输入，而不限制在同一单元的输出到输入。

【例】：执行以下指令序列

```
DADD   R1,R2,R3
LD     R4,0(R1)
SD     R4,12(R1)
```


使用直通技术解决数据冒险引起的停顿

Load的结果通过MEM/WB流水线寄存器从存储器输出端直通到用于保存它的store指令的存储器输入端

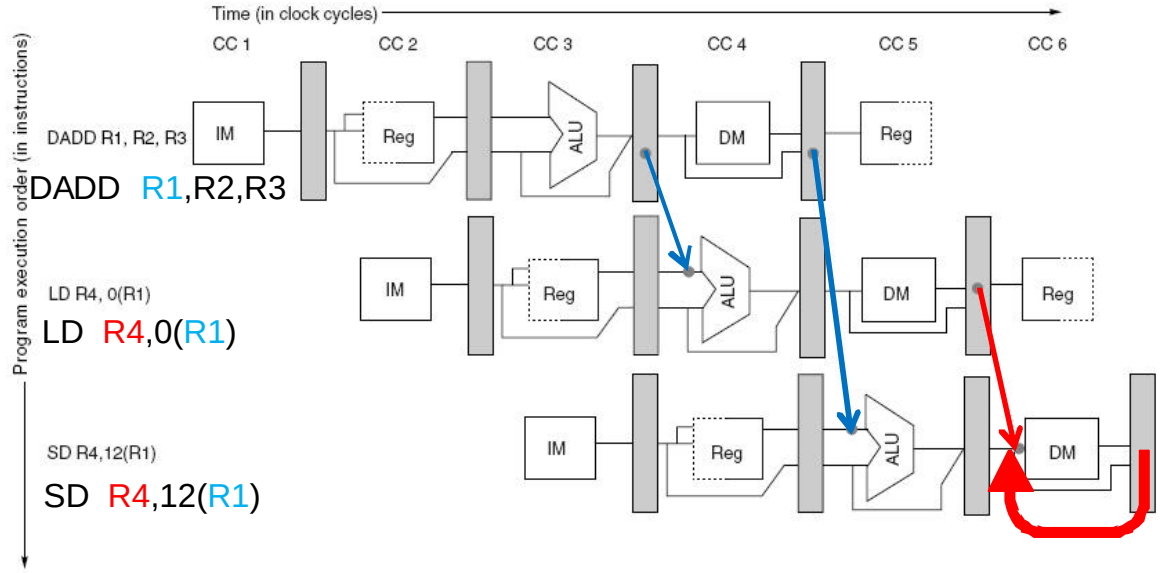


Figure A.8 Forwarding of operand required by stores during MEM. The result of the load is forwarded from the memory output to the memory input to be stored. In addition, the ALU output is forwarded to the ALU input for the address calculation of both the load and the store (this is no different than forwarding to another ALU operation). If the store depended on an immediately preceding ALU operation (not shown above), the result would need to be forwarded to prevent a stall.

直通技术的实现原理

要直通到ALU，需要为每个ALU输入端增加选择输入通路，分别来自于：

（1）EX段末尾的ALU输出端；

（2）MEM段末尾的ALU输出端；

（3）MEM段末尾的存储器输出端；

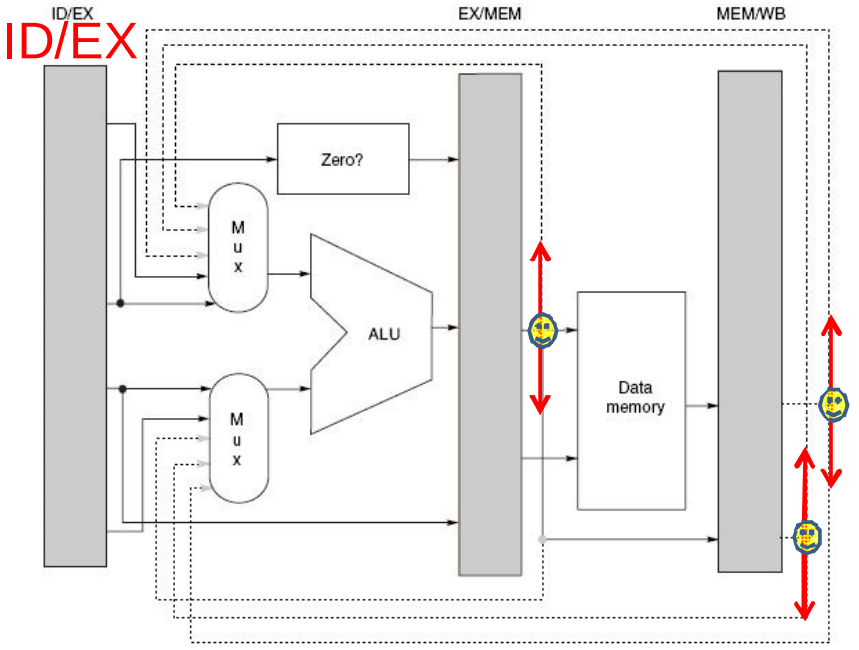


Figure A.23 Forwarding of results to the ALU requires the addition of three extra inputs on each ALU multiplexer and the addition of three paths to the new inputs. The paths correspond to a bypass of (1) the ALU output at the end of the EX, (2) the ALU output at the end of the MEM stage, and (3) the memory output at the end of the MEM stage.

直通技术的实现原理

数据冒险（数据相关性）检测方法

参考前图，针对ALU操作的数据相关，可以分为两类：

1. ALU的操作输入，需要前面一条指令的执行结果，当前其保存在EX/MEM流水线寄存器中；
2. ALU的操作输入，需要前面第二条指令的执行结果，当前其保存在MEM/WB流水线寄存器中。

数据冒险（数据相关性）检测方法

与上一条指令相关的检测方法

```
DAAD   R1,R2,R3     ; [R1] ← [R2] + [R3]
DSUB   R4,R1,R5     ; [R4] ← [R1] - [R5]
AND    R6,R1,R7     ; [R6] ← [R1] & [R7]
```

指令格式

```
DAAD   Rd,Rs,Rt     ; [Rd] ← [Rs] + [Rt]
DSUB   Rd,Rs,Rt     ; [Rd] ← [Rs] - [Rt]
AND    Rd,Rs,Rt     ; [Rd] ← [Rs] & [Rt]
```

检测方法

```
ID/EX.RegisterRs = EX/MEM.RegisterRd
ID/EX.RegisterRt = EX/MEM.RegisterRd
```


数据冒险（数据相关性）检测方法

与上第二条指令相关的检测方法

```
DAAD   R1,R2,R3     ; [R1] ← [R2] + [R3]
DSUB   R4,R1,R5     ; [R4] ← [R1] - [R5]
AND    R6,R1,R7     ; [R6] ← [R1] & [R7]
```

指令格式

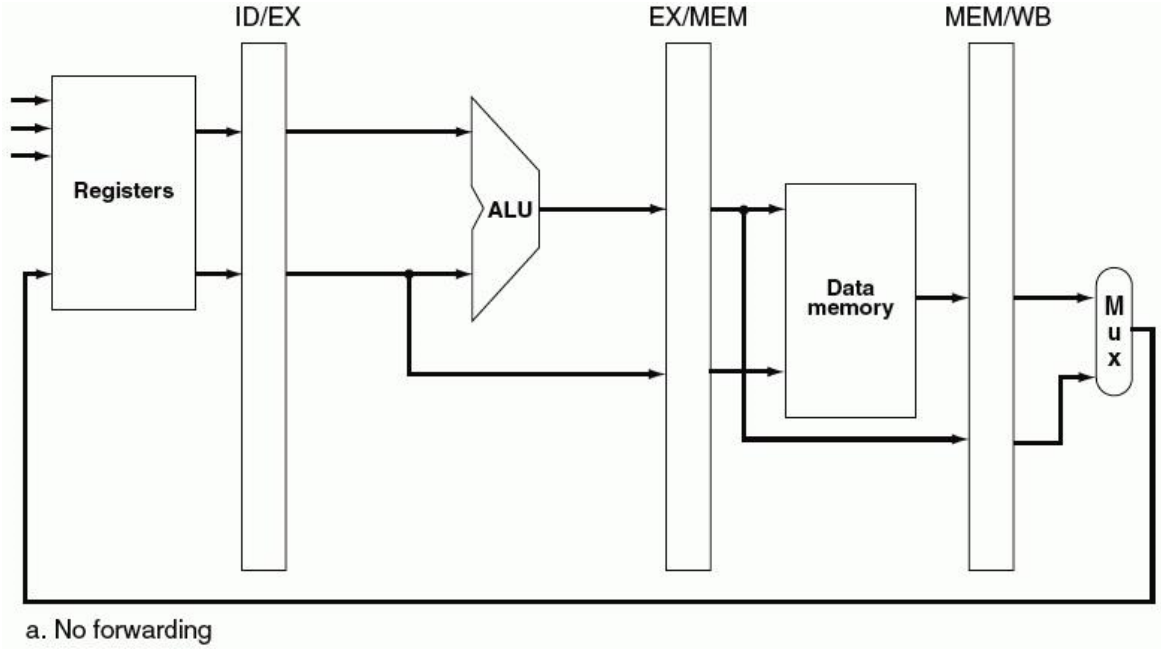
```
DAAD   Rd,Rs,Rt     ; [Rd] ← [Rs] + [Rt]
DSUB   Rd,Rs,Rt     ; [Rd] ← [Rs] - [Rt]
AND    Rd,Rs,Rt     ; [Rd] ← [Rs] & [Rt]
```

检测方法

```
ID/EX.RegisterRs = MEM/WB.RegisterRd
ID/EX.RegisterRt = MEM/WB.RegisterRd
```

直通技术的实现原理

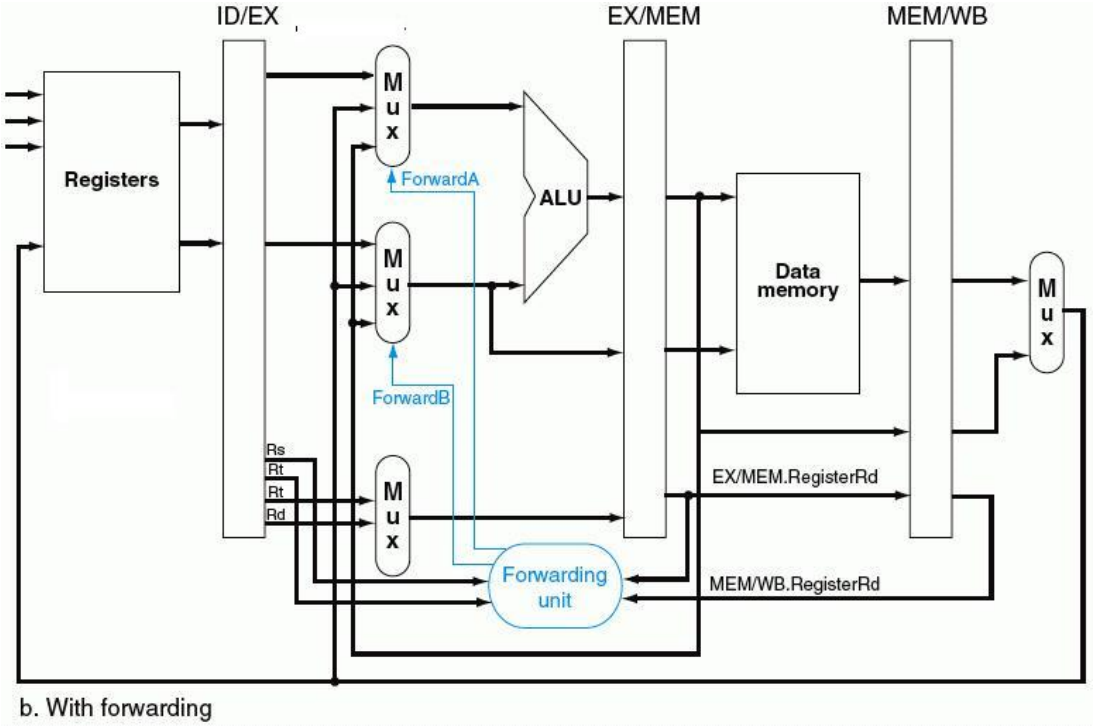
没有直通旁路的数据流图



a. No forwarding

直通技术的实现原理

加入了转发直通旁路的数据流图



b. With forwarding

直通技术的实现原理

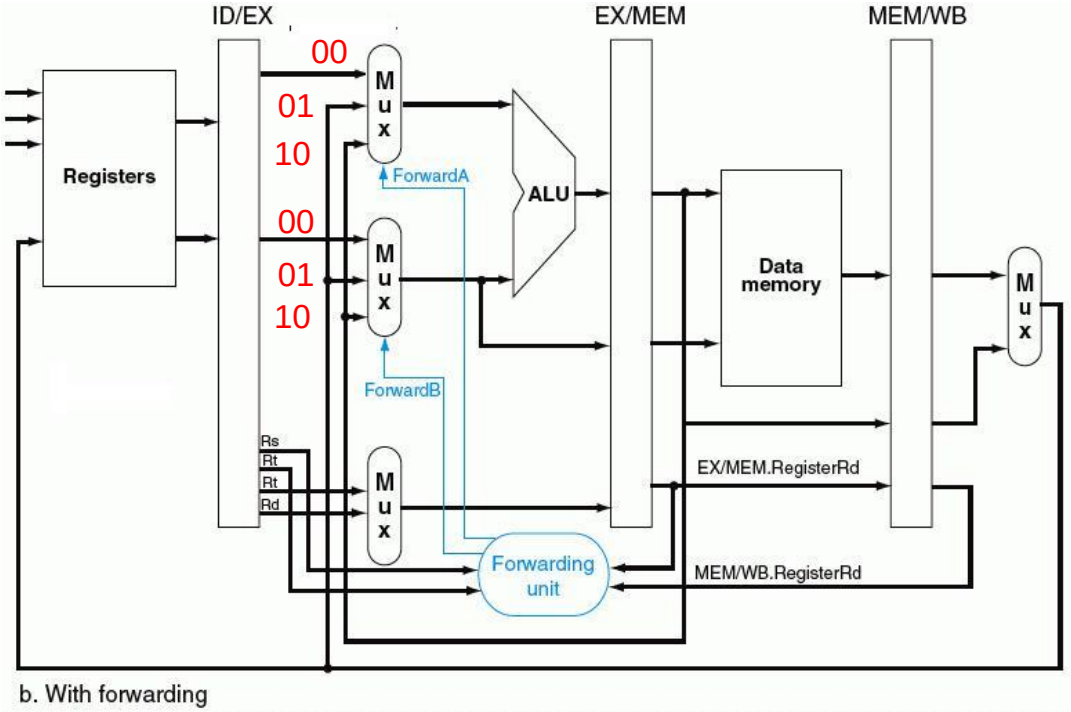
前图中多路选择器的控制信号值

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

FIGURE 6.31 The control values for the forwarding multiplexors in Figure 6.30. The signed immediate that is another input to the ALU is described in the elaboration at the end of this section.

直通技术的实现原理

加入了转发直通旁路的数据流图



b. With forwarding

数据冒险（数据相关性）检测方法

因为有特殊情况

1.由于有一些指令不写寄存器，所以可能导致一些不必要的转发；

2. 对0号寄存器（总是0）要特别对待。

因此，前述判断方法要修正。

数据冒险（数据相关性）检测方法

修正与上一条指令相关的检测方法

DAAD R1,R2,R3 ；[R1] ← [R2] + [R3]

DSUB R4,R1,R5 ；[R4] ← [R1] - [R5]

AND R6,R1,R7 ；[R6] ← [R1] & [R7]

检测方法

ID/EX.RegisterRs = EX/MEM.RegisterRd

ID/EX.RegisterRt = EX/MEM.RegisterRd

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
```

数据冒险（数据相关性）检测方法

修正与上第二条指令相关的检测方法

```
DAAD   R1,R2,R3     ; [R1] ← [R2] + [R3]
DSUB   R4,R1,R5     ; [R4] ← [R1] - [R5]
AND    R6,R1,R7     ; [R6] ← [R1] & [R7]
```

检测方法

ID/EX.RegisterRs = MEM/WB.RegisterRd

ID/EX.RegisterRt = MEM/WB.RegisterRd

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```


数据冒险（数据相关性）检测方法

针对以下情况，第二种判断标准仍需做修正

```
DADD   R1,R1,R2
DADD   R1,R1,R3
DADD   R1,R1,R4   ……
```

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```

通过直通转发解决数据冒险的数据通路

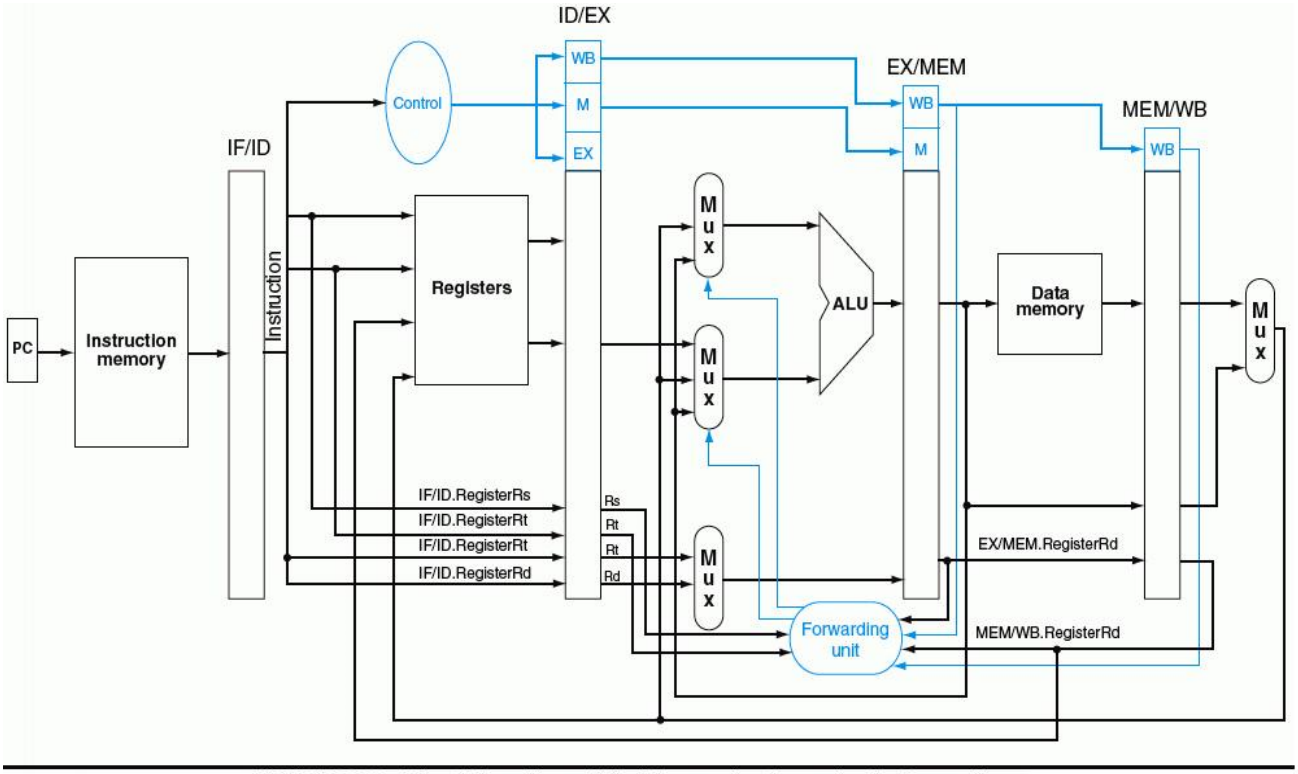


FIGURE 6.32 The datapath modified to resolve hazards via forwarding.

通过直通转发技术解决数据冒险

似乎数据冲突

都可以通过直通旁路技术

解决掉？

需要停顿的数据冒险

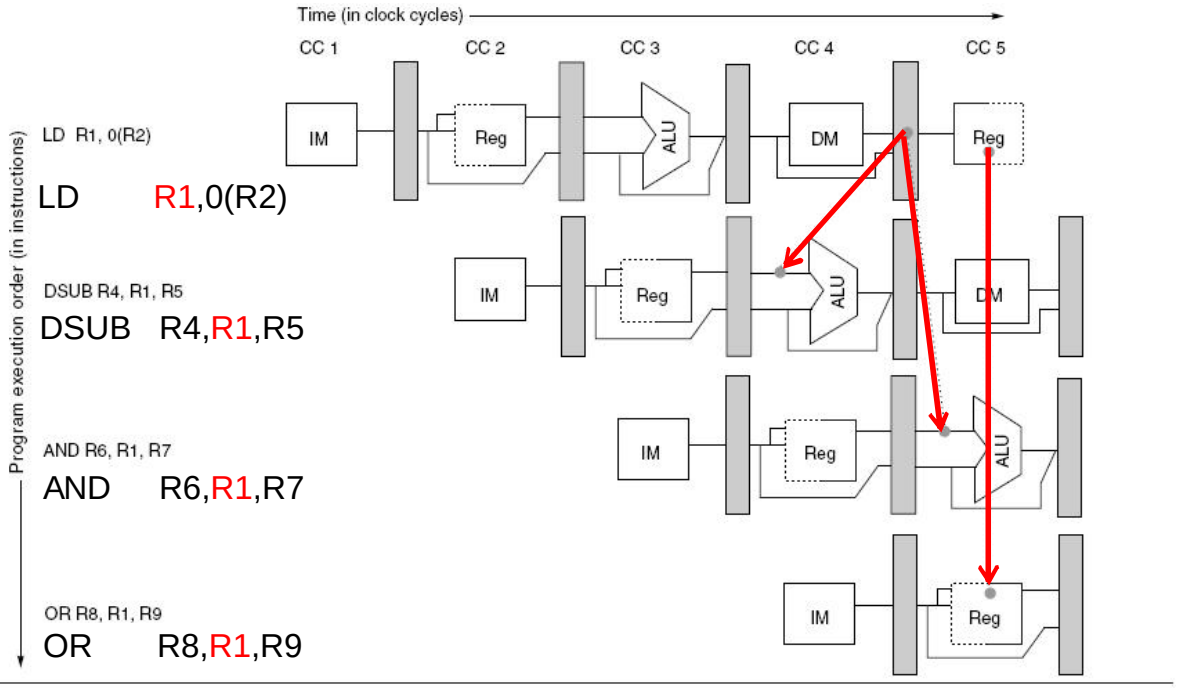
很遗憾，并非所有的数据冒险都可以采用旁路技术解决

【例】有如下指令序列：

```
LD      R1,0(R2)
DSUB    R4,R1,R5
AND     R6,R1,R7
OR      R8,R1,R9
```


需要停顿的数据冒险

Load指令相关所引起的停顿



LD R1,0(R2)

DSUB R4,R1,R5

AND R6,R1,R7

OR R8,R1,R9

Figure A.9 The load instruction can bypass its results to the AND and OR instructions, but not to the DSUB, since that would mean forwarding the result in "negative time."

对于load指令所导致的冒险，不得不加入停顿

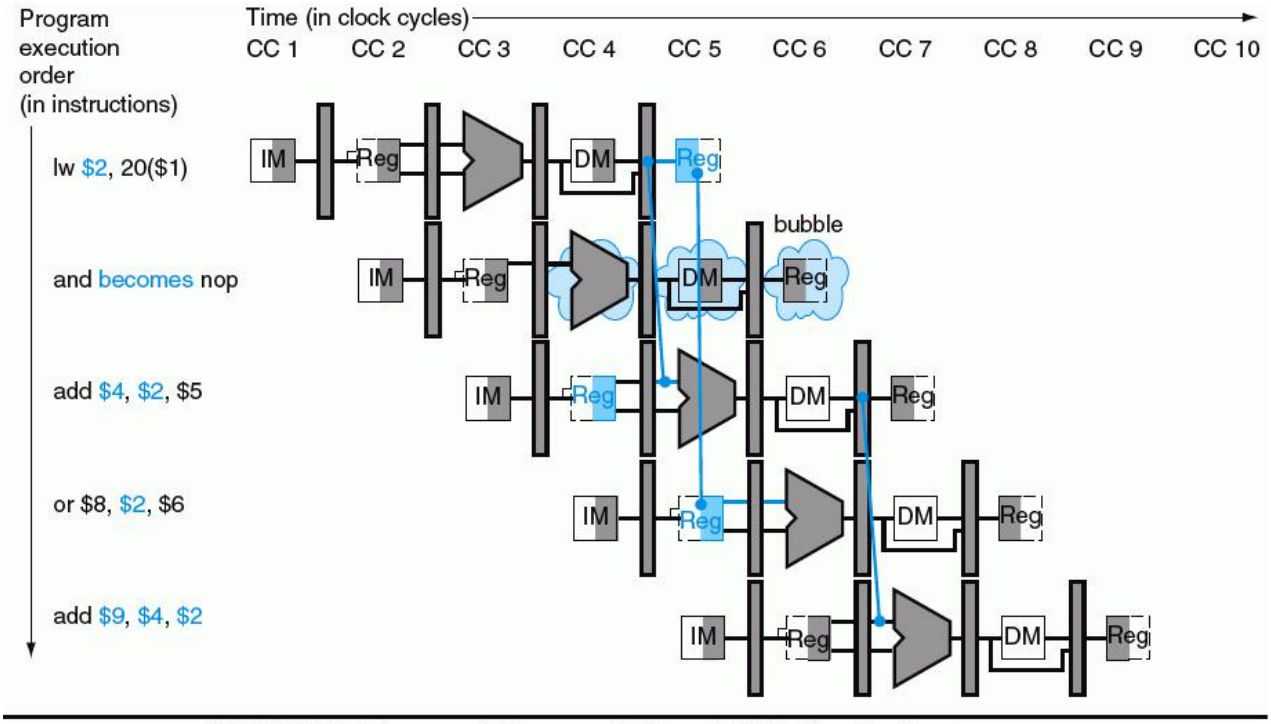


FIGURE 6.35 The way stalls are really inserted into the pipeline.

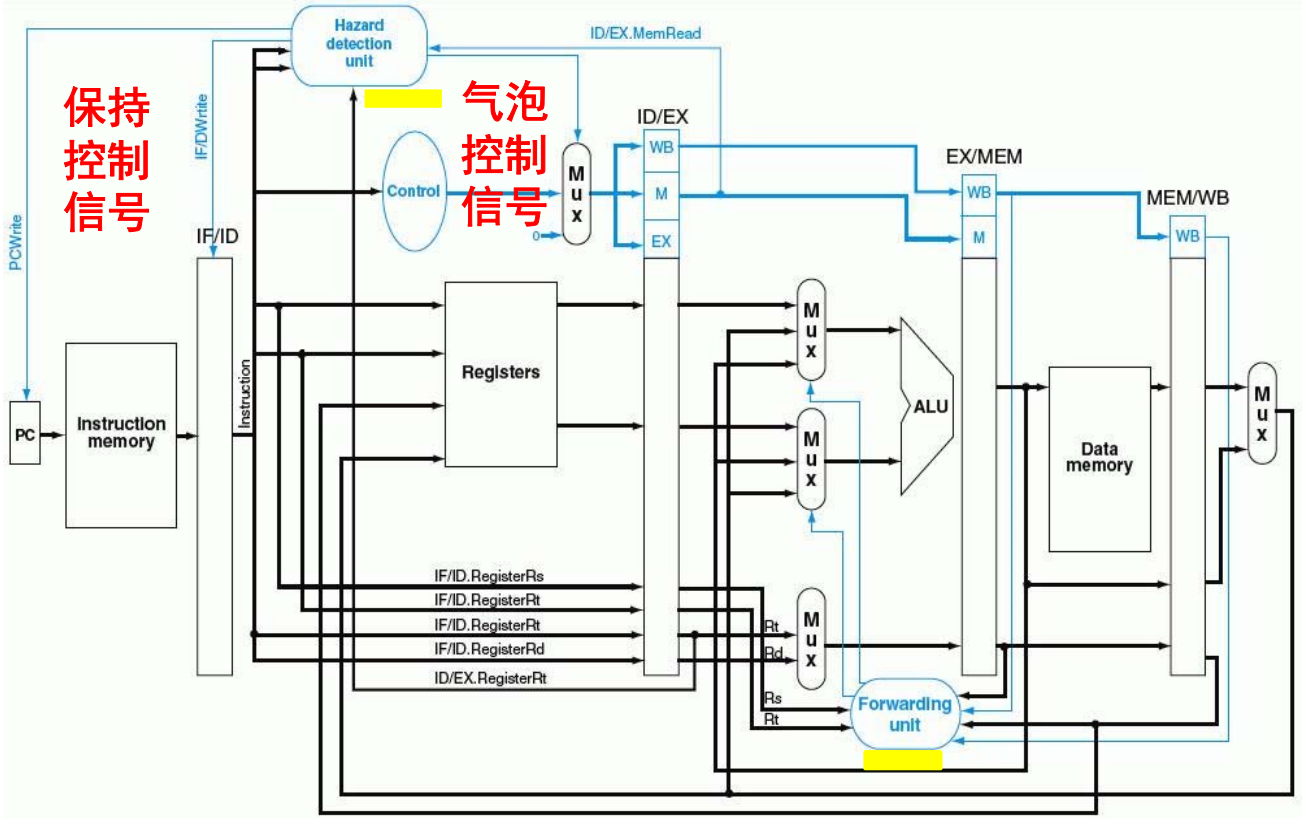
Load指令所导致的数据冒险（数据相关性）检测方法

```
if (ID/EX.MemRead and
   ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
     (ID/EX.RegisterRt = IF/ID.RegisterRt)))
      stall the pipeline
```

如何实现插入气泡/停顿

1. 对后续指令，防止其继续流动，需要保持PC值和IF/ID流水线寄存器值不发生变化。
2. 对当前停顿流水段，通过将所有EX、MEM和WB阶段的所有9个控制信号置零/无效，使该流水步在后续推进中不产生任何动作，从而产生一条NOP指令/气泡。

具有直通和停顿2类数据冒险检测和处理的流水线



流水线的主要障碍——流水线冒险

流水线冒险（Hazards）

1. 结构冒险：也叫结构冲突，当硬件在指令重叠执行中不能支持指令所有可能的组合时，所发生的资源冒险。
2. 数据冒险：在同时执行的几条指令中，一条指令依赖于前一条指令的结果数据，却得不到时，发生的冒险。
3. 控制冒险：流水线中的转移指令或其它改写PC的指令所造成的冒险。

■ 流水线中的“冒险”，可能会引起流水线的“停顿”，从而影响流水线的性能。

4.8 Control Hazards

控制冒险（control hazard / branch hazard）

- 由于转移指令的执行导致指令流（PC）发生变化，原来已进入流水线的指令可能作废，从而导致流水线性能损失。
- 在MIPS流水线中，控制冒险所造成的损失，比数据冒险更大。

4.8 Control Hazards

控制冒险（control hazard / branch hazard）

- Also called branch hazard. An occurrence in which the proper instruction cannot execute in the proper clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction address is not what the pipeline expected.

- Arise from the pipelining of branches and other instructions that change the PC.

减少流水线的转移代价

几种简单的编译时调度方法：

1. 最简单的方法：冻结或冲刷流水线。
2. 对所有的转移都按未选中处理。
3. 预测转移被选中。
4. 采用转移延迟（添加延迟槽branch delay slot）。

通过把判零检测和地址计算前移到ID段来缩短转移冒险引起的停顿

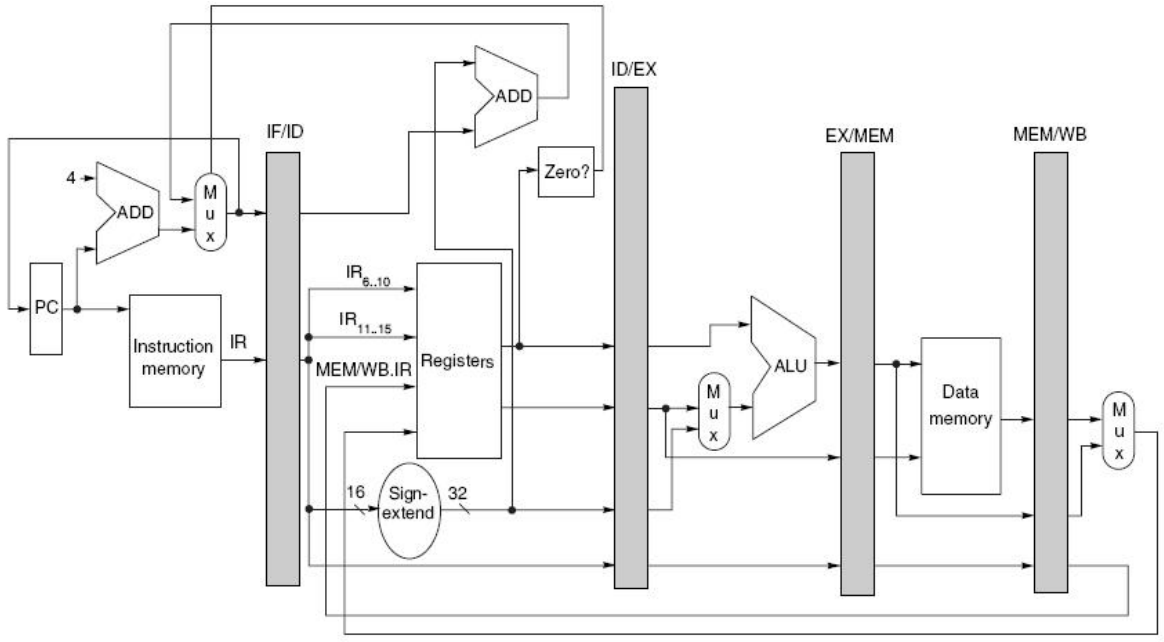


Figure A.24 The stall from branch hazards can be reduced by moving the zero test and branch-target calculation into the ID phase of the pipeline.

减少流水线的转移代价——调度转移延迟槽

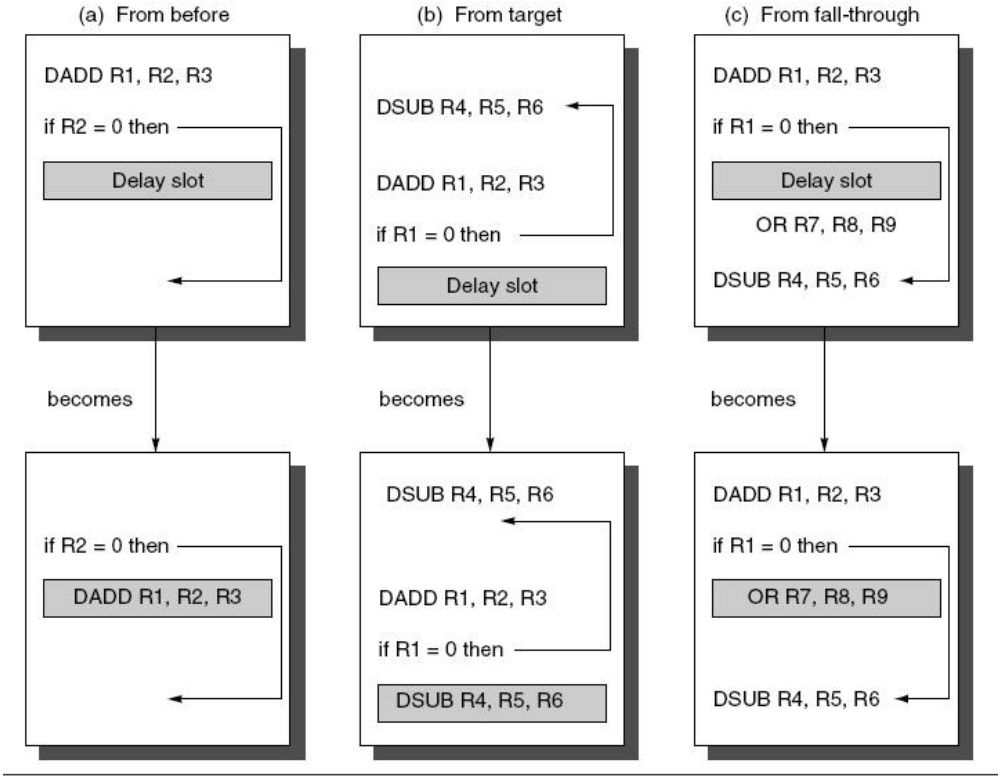


Figure A.14 Scheduling the branch delay slot.

减少流水线的转移代价——动态分支预测

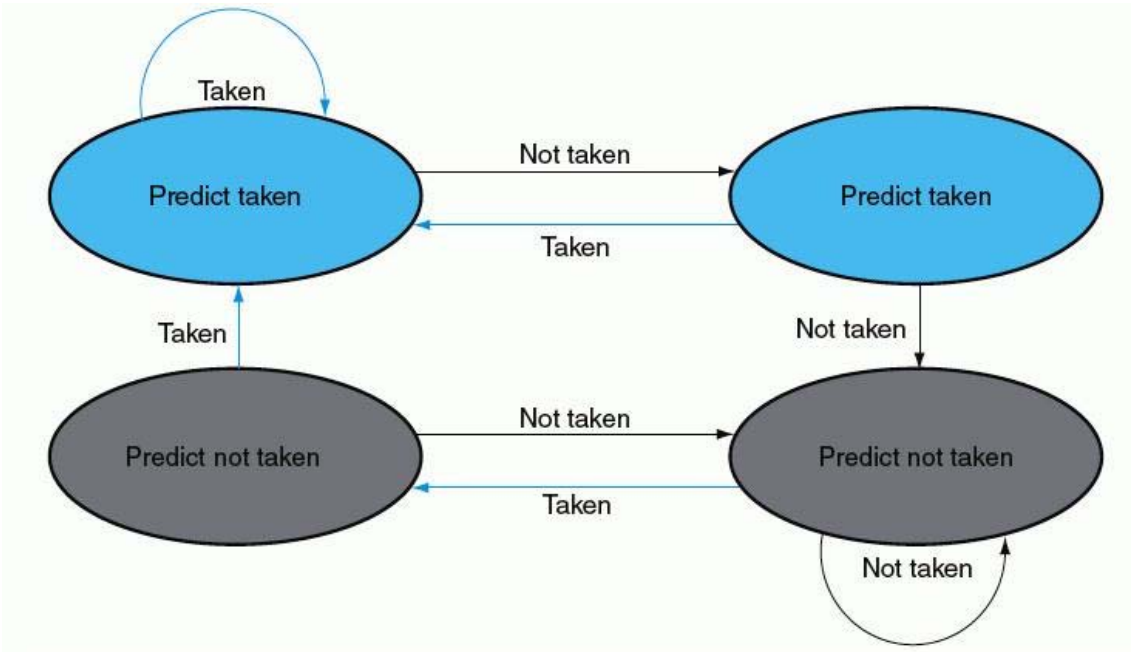


FIGURE 6.39 The states in a 2-bit prediction scheme. By using 2 bits rather than 1, a branch that strongly favors taken or not taken—as many branches do—will be mispredicted only once. The 2 bits are used to encode the four states in the system. The two-bit scheme is a general instance of a counter-based predictor, which is incremented when the prediction is accurate and decremented otherwise, and uses the midpoint of its range as the division between taken and not taken.

流水线的主要障碍——流水线冒险

具有冒险检测和处理能力的流水线

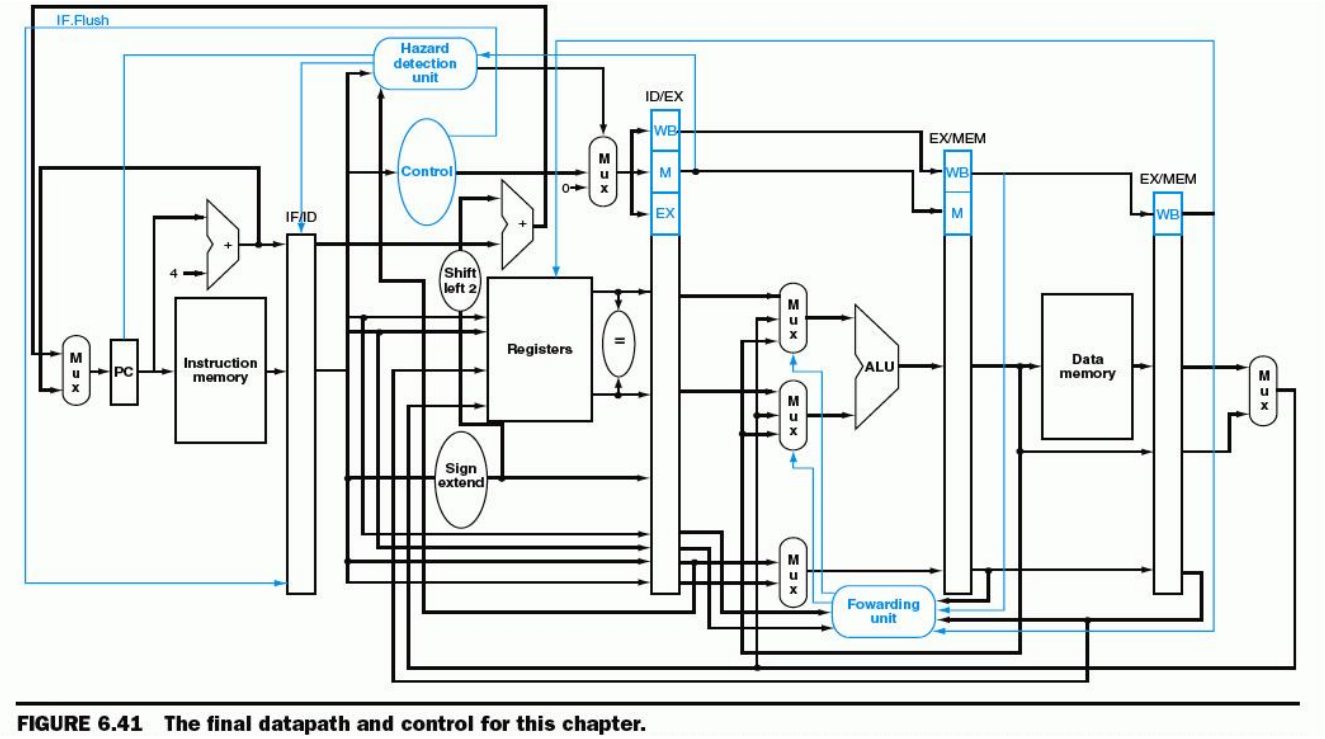


FIGURE 6.41 The final datapath and control for this chapter.

4.8 Control Hazards

Check Yourself

predict

Consider three branch prediction schemes: branch not taken, predict taken, and dynamic prediction. Assume that they all have zero penalty when they predict correctly and two cycles when they are wrong. Assume that the average predict accuracy of the dynamic predictor is 90%. Which predictor is the best choice for the following branches?

1. A branch that is taken with 5% frequency
2. A branch that is taken with 95% frequency
3. A branch that is taken with 70% frequency

课程小结

流水线原理部分小结

1. 单周期数据通路
2. 流水线寄存器
3. 流水线冒险
4. 数据直通旁路
5. 如何检测数据冒险
6. 如何插入气泡（停顿）
7. 分支预测
8. 具有冒险检测和处理能力的简单完备流水线

20条实验用指令

```
add  rd, rs, rt   ; rd <-- rs + rt
sub  rd, rs, rt   ; rd <-- rs - rt
and  rd, rs, rt   ; rd <-- rs & rt
or   rd, rs, rt   ; rd <-- rs | rt
xor  rd, rs, rt   ; rd <-- rs ^ rt
sll  rd, rt, sa   ; rd <-- rt << sa
srl  rd, rt, sa   ; rd <-- rt >> sa (logical)
sra  rd, st, sa   ; rd <-- rt >> sa (arithmetic)
jr   rs           ; PC <-- rs
addi rt, rs, imm  ; rt <-- rs + (sign)imm
andi rt, rs, imm  ; rt <-- rs & (zero)imm
ori  rt, rs, imm  ; rt <-- rs | (zero)imm
xori rt, rs, imm  ; rt <-- rs ^ (zero)imm
lw   rt, imm(rs)  ; rt <-- memory[rs + (sign)imm]
sw   rt, imm(rs)  ; memory[rs + (sign)imm] <-- rt
beq  rs, rt, imm  ; if (rs == rt) PC <-- PC + 4 + (sign)imm << 2
bne  rs, rt, imm  ; if (rs != rt) PC <-- PC + 4 + (sign)imm << 2
lui  rt, imm      ; rt <-- imm << 16
j    addr         ; PC <-- (PC+4)[31..28],addr<<2
jal  addr         ; $31 <-- PC+8; PC <-- (PC+4)[31..28],addr << 2
```

注意: MIPS 中 jal 指令保存的返回地址是 PC+8 (延遲轉移)。

流水线的各种相关

Hazard — 妨礙正常流水的各種狀況

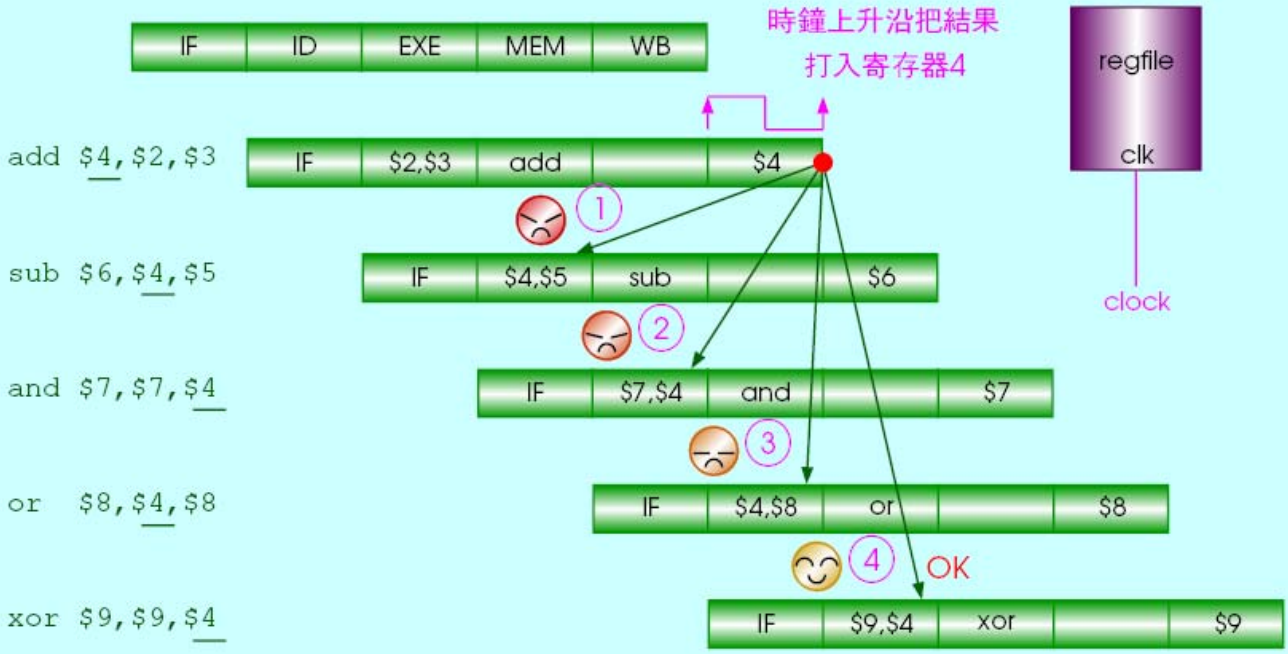
1. 構造相關 (Structural Dependency):
   - 由于硬件資源短缺而造成流水綫堵塞
2. 控制相關 (Control Dependency):
   - 轉移或跳轉指令或中斷等改變指令流水時發生
3. 數據相關 (Data Dependency):
   - 一條指令使用它的上一條指令的執行結果

“相关”问题的解决方法

1. 構造 (Structural Dependency):
   - 增加硬件資源
   - 編譯器調度
2. 控制相關 (Control Dependency):
   - 延遲轉移
   - 根據轉移預測而投機執行
3. 數據相關 (Data Dependency):
   - 編譯器調度
   - 內部前推 (Internal Forwarding, Bypass)

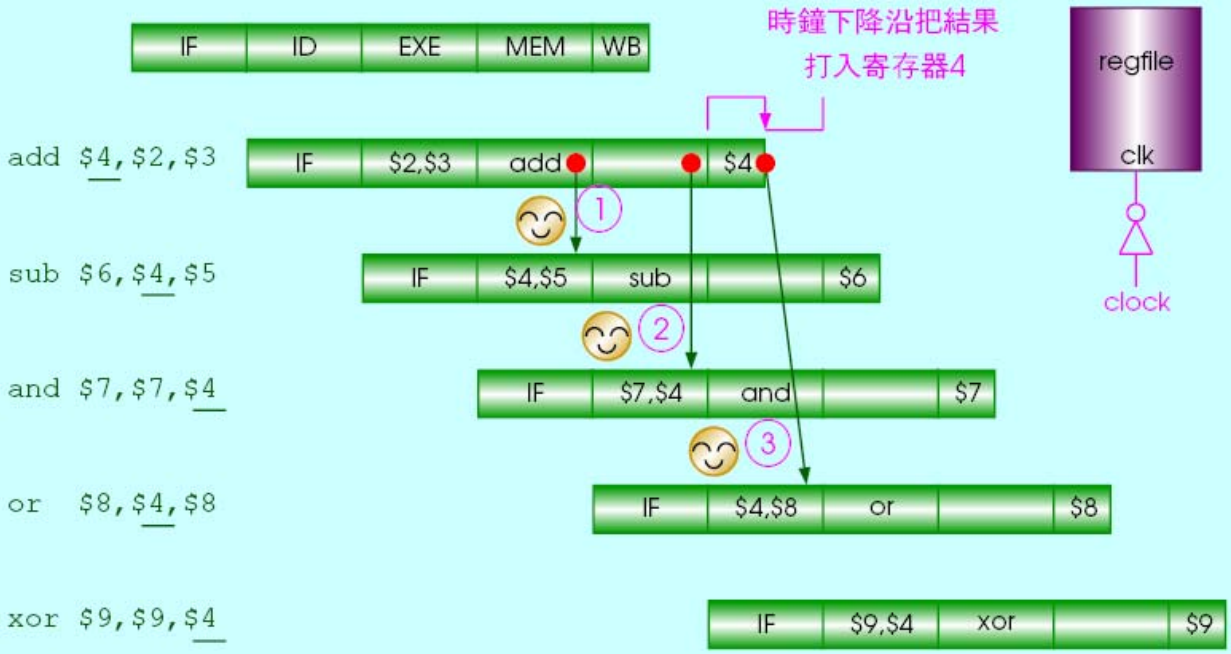
数据相关（Data Dependency）

數據相關: 一條指令使用它的上一條指令的執行結果

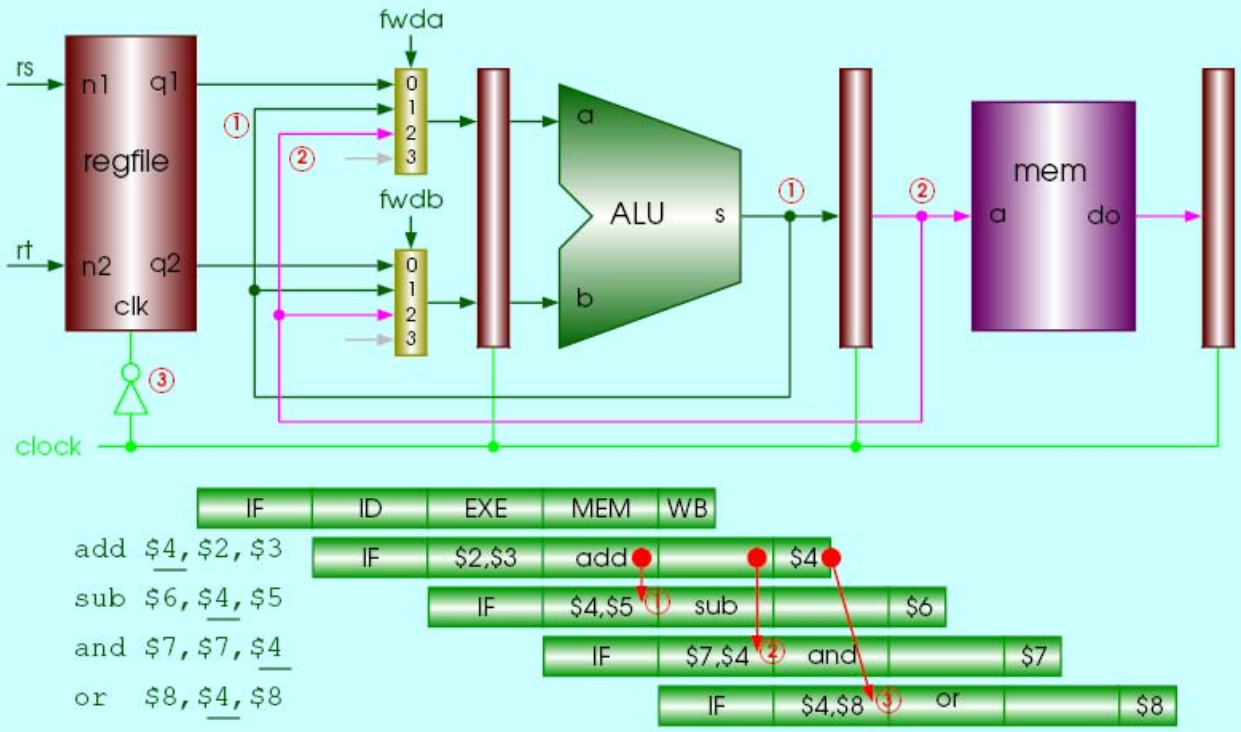


内部前推（Forwarding）

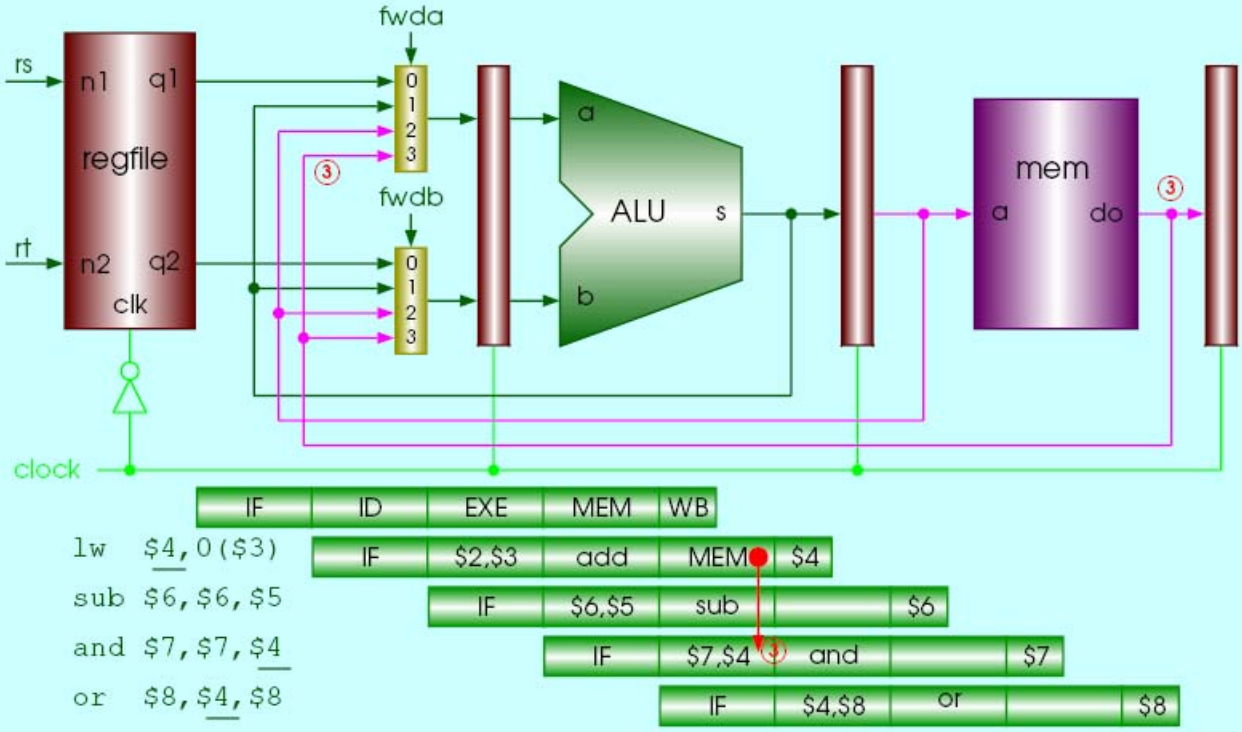
增加內部前推電路并在時鐘下降沿打入結果(一個周期的一半處)



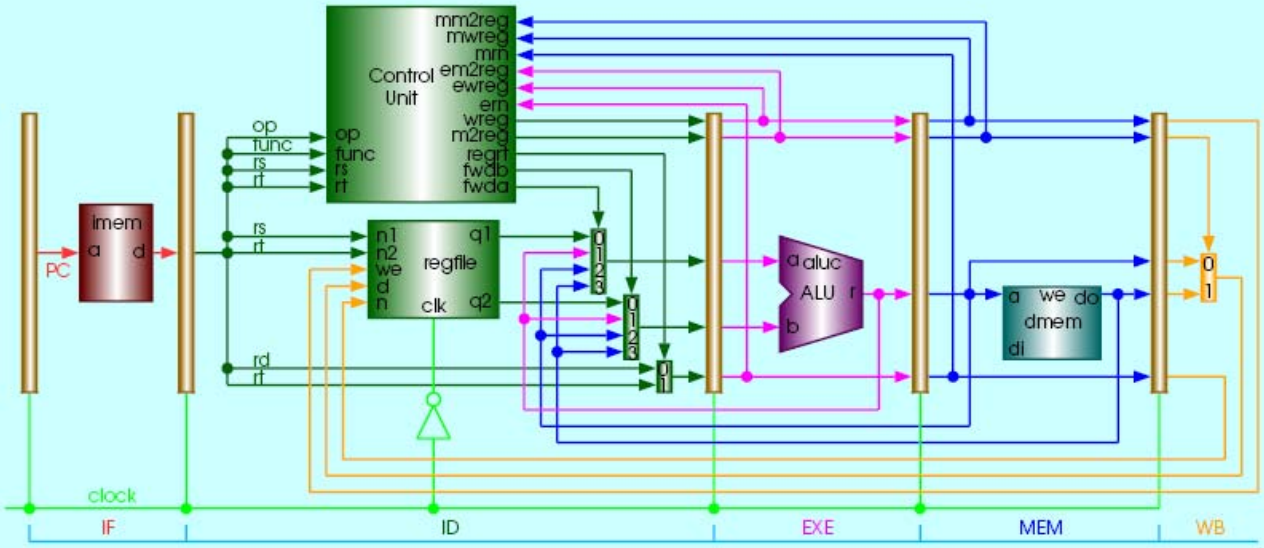
内部前推（Forwarding）—from add指令



内部前推（Forwarding）—from lw指令

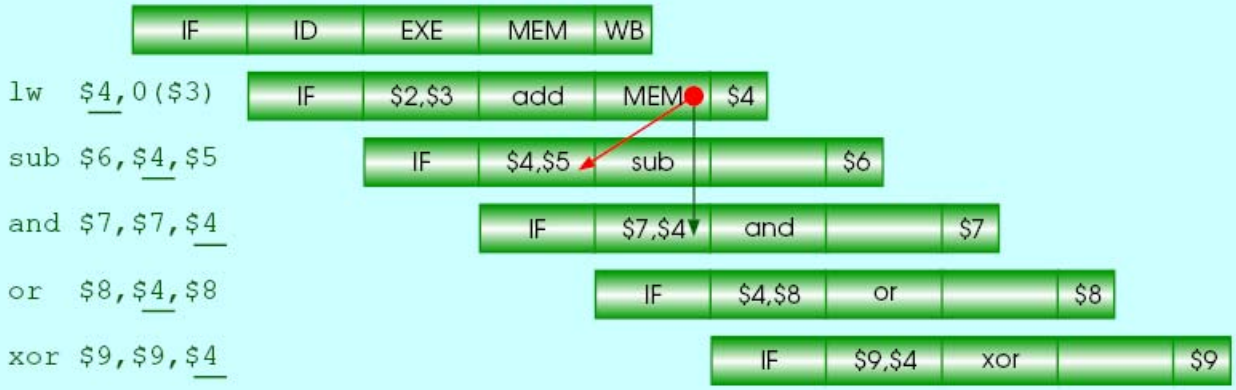


内部前推（Forwarding）的三个来源



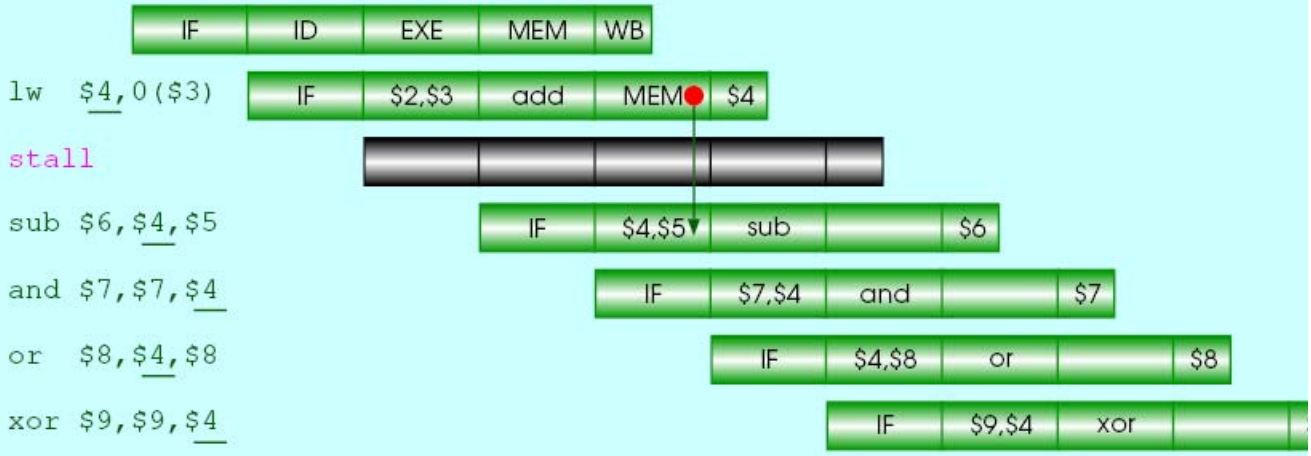
1. Forward ALU result from EXE stage to ID stage
2. Forward ALU result from MEM stage to ID stage
3. Forward MEM result from MEM stage to ID stage

要前推lw指令取来的数据？



可以把 lw 指令取來的數據前推給 and, 如果 sub 也提出前推的要求就太過分了: 你要做減法, 但需要的數據還沒從存儲器中取出來呢, 還是等一等吧

等一等：流水管道出一气泡，还是黑颜色的



流水綫要暫停一個周期。方法是不往PC和IR中寫新的數據，因此需要一個wpcir信號

事实上，Load相关延迟一个周期后，仍需要进行从MEM阶段向前的“forwarding”。（因此，ALU事实上共有4个源）

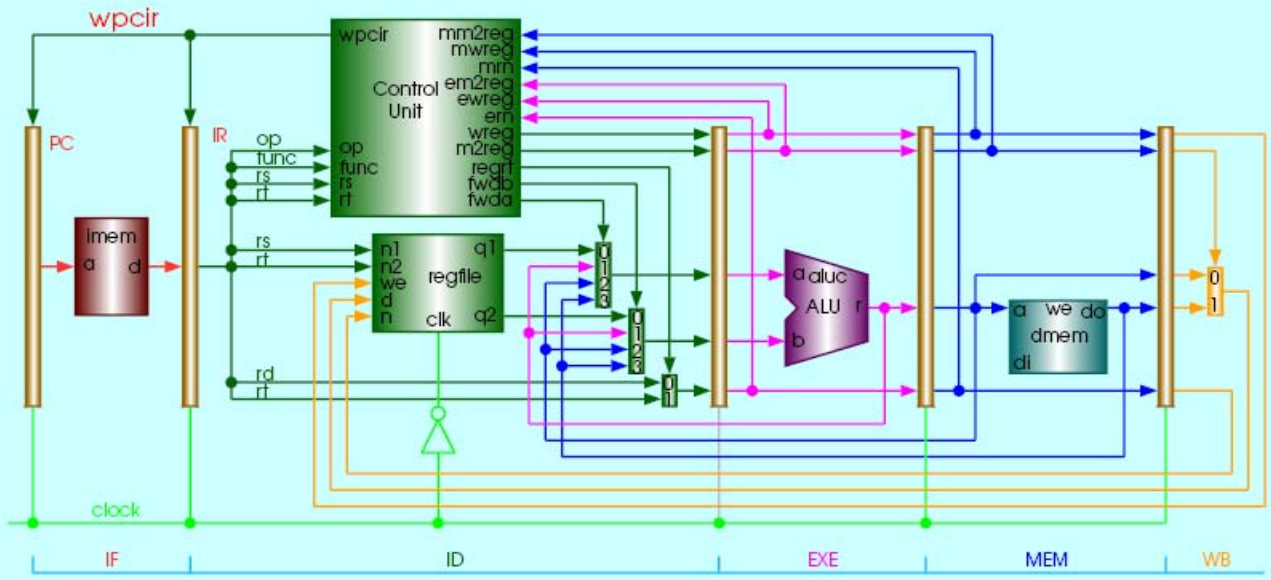
内部推进选通信号 fwda

```
fwda = 2'b00; // default forward a: no hazards
if (ewreg & (ern != 0) & (ern == rs) & ~em2reg) begin
    fwda = 2'b01; // select exe_alu
end else begin
    if (mwreg & (mrn != 0) & (mrn == rs) & ~mm2reg) begin
        fwda = 2'b10; // select mem_alu
    end else begin
        if (mwreg & (mrn != 0) & (mrn == rs) & mm2reg) begin
            fwda = 2'b11; // select mem_lw
        end
    end
end
```

内部推进选通信号 fwdb

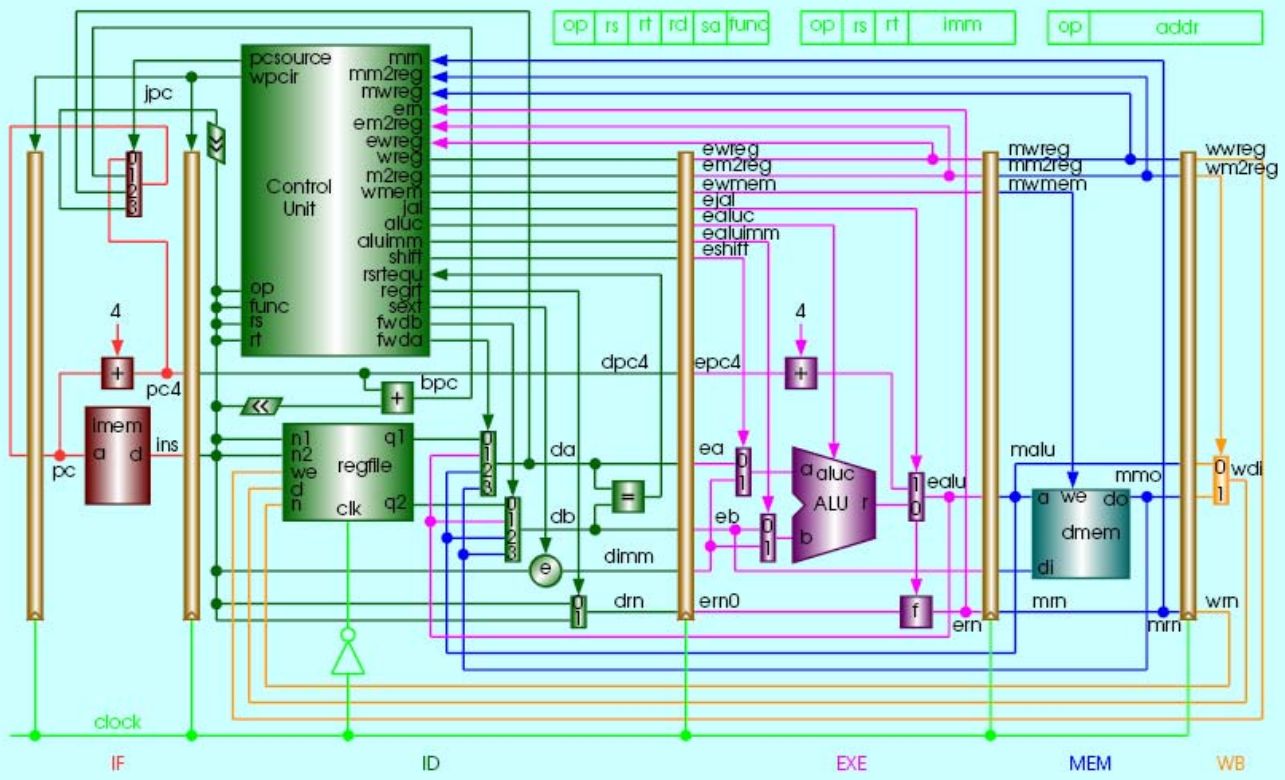
```
fwdb = 2'b00; // default forward b: no hazards
if (ewreg & (ern != 0) & (ern == rt) & ~em2reg) begin
    fwdb = 2'b01; // select exe_alu
end else begin
    if (mwreg & (mrn != 0) & (mrn == rt) & ~mm2reg) begin
        fwdb = 2'b10; // select mem_alu
    end else begin
        if (mwreg & (mrn != 0) & (mrn == rt) & mm2reg) begin
            fwdb = 2'b11; // select mem_lw
        end
    end
end
```


流水线暂停



wpcir 爲 0 時, 不寫 PC 和 IR

流水线CPU



Acknowledgements

- These slides contain material developed and copyright by:
  - —Arvind (MIT)
  - —Krste Asanovic (MIT/UCB)
  - —John Kubiatowicz (UCB)
  - —David Patterson (UCB)
  - —Geng Wang (SJTU)
  - —Yanmin Zhu (SJTU)
  - —Li Yamin(Hosei Univ.)