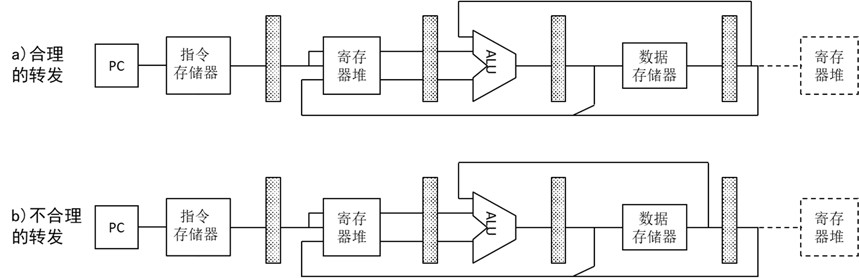
1. 为了将 DM 结果向 ALU 转发，本章采用如图 6-16(a)的思路：从最后一级流水线寄存器

MEM/WB 向 ALU 转发。但这个设计在执行如下指令序列时必须暂停一个时钟周期。设计师认为采用如图 6-16(b)的思路就可以解决这个问题：从直接从 DM 向 ALU 转发。虽然图 6-16(b)可以解决上述问题，但却使得流水线时钟频率下降了，请分析具体原因

（假设 IM 读出、RF 读出、ALU、DM 读出的延迟均为 L）。

lw $1, xxx add yyy, $1, zzz



# 图 6-16 DM 转发的两种思路

1. 在流水中插入寄存器可以提高时钟频率。但是，随着级数增多，流水线性能提升会遇到瓶颈。首先，寄存器自身的时序开销（如寄存器建立时间与保持时间等）对于性能改善的影响越来越大。其次，随着流水线级数的增长，分支冒险会导致流水线排空的问题愈发严重。第三，数据冒险造成的暂停也会随之增多。

假设：5 级流水线 CPI 为 1.2，且每增加 1 级，CPI 增加 0.1；单周期 CPU 关键路径延迟为 800ps，寄存器自身时序开销为 50ps。

* 1. 建立 CPI 与流水线级数 N（N≥5）的计算公式。
  2. 建立时钟周期延迟 Tc 与流水线级数 N 的计算公式。
  3. 给出一条指令执行时间的计算公式。
  4. 请指出 N 为多少时，流水线性能最好。
  5. 请指出 N 为多少时，流水线性能改善最为显著。

1. 设计师将单周期数据通路改造为如图 6-17 所示的 3 级流水线。假设 RF 不支持内部转发。

ALU

寄存

器堆

数据

存储器

单周期

数据通路

流水线

数据通路

指令

存储器

寄存

器堆



PC

ALU

寄存

器堆

数据

存储器

指令

存储器

寄存

器堆

PC

取指令

译码

/

读操作数

/

执行

访存

/

回写

# 图 6-17 改造单周期数据通路为 3 级流水线

1. 流水线是否仍然可能会因为 beq 而需要清空流水线？如果会，最多有几条指令会

被清除？

1. 假设指令集只有{lw，add}，请以 rs 寄存器为例，增加旁路以应对所有的数据冒险

可能。给出思路即可，不用讨论因此带来的 MUX 及其控制。

1. 对于第 2 问的指令集，能否消除 rs 寄存器相关的全部数据冒险？为什么？
2. 同样是从 DM 向 ALU 的转发，图 6-18(a)的设计会使得 5 级流水线性能下降。请分析图

6-18(b)的设计会导致 3 级流水线性能下降吗？假设 IM 读出、RF 读出、ALU、DM 读出的延迟均为 L，忽略所有控制器延迟及 MUX 延迟。

ALU

寄存

器堆

数据

存储器

指令

存储器

寄存

器堆



PC

5级流水线：不合理的转发

ALU

寄存

器堆

数据

存储器

指令

存储器

寄存

器堆



PC

3级流水线：合理的转发

图 6-18 从 DM 向 ALU 的转发

1. 下图所示的流水线 CPU 执行如下指令序列。

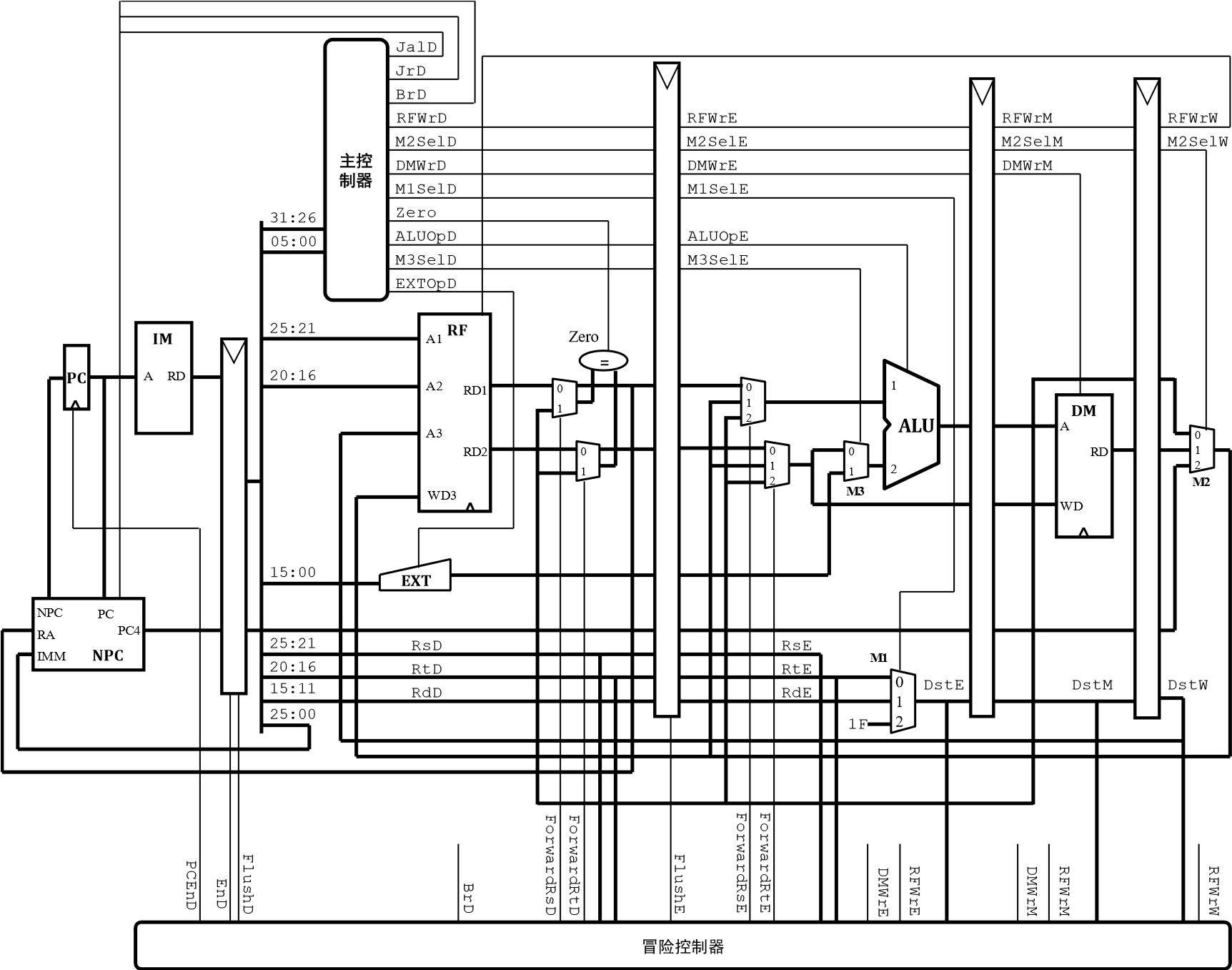
I1 lw $1, 0($2)

I2 addi $1, $1, $1

I3 sw $1, 0($2) I4 lw $1, 4($2)

I5 sw $1, 8($2)

* 1. 分析上述指令执行过程中流水线共计需要暂停多少个时钟周期。
  2. 是否可以增加转发来提升流水线性能？如果可以，请简述设计思路。



1. 在第 5 题图中所示的流水线 CPU 执行某程序，其指令分布如下：load 占 15%，store 为 10%，分支指令为 10%，R 型计算类指令为 65%。假设：load-R 导致暂停概率为 30%； load-store 导致的暂停概率为 5%；分支指令预测成功率为 75%。计算流水线执行该程

序的 CPI。

1. 如图 6-19 所示，某 MIPS 标准 5 级流水线仅支持 M 级向 D 级的转发（注意：寄存器堆无内部转发）。某程序员编写了如下 MIPS 代码，请回答下列问题。

图

6-19

仅支持

M

级向

D

级转发的

5

级流水线

指令

存储器

A

RD

**PC**

寄存器堆

A1

A2

A3

WD

RD1

RD2

**ALU**

Zero

ALU

结果

数据

存储器

A RD

WD

25:21

20:16

15:11

**+**

4

0

1

E

M

W

D

扩展

15:00

0

1

0

1

0

1

I1: lw $1, 0($2)

I2: sw $1, 0($1)

I3: add $3, $2, $2

I4: sub $4, $1, $3

I5: or $5, $5, $6

* 1. 请指出上述指令片段在上述流水线中执行时存在的所有数据相关。
  2. 请通过调整指令顺序来优化上述指令片段以最大化减少暂停。
  3. 对于优化前和优化后指令片段，分别给出流水线的执行时间，并说明理由。示例：

对于 2 条无冒险的指令片段，则流水线执行时间为 6 个时钟周期。

答案

1. 由于从 DM 向 ALU 反馈，使得从 EX/MEM 寄存器到其自身之间存在了组合逻辑，即

DM+ALU，其延迟为 2L。流水线频率变为 1/2L，因此图 b 流水线性能是图 a 的 50%。

2.

2. 一条指令执行时间
3. N = 10，性能最好。
4. N = 6，性能改善最为显著。

252

238

230

225

222

221

221

222

223

225

227

%

5

%

3

2

%

1

%

%

1

0

%

0

%

%

-1

%

-1

%

-1

%

-2

%

-1

0

%

1

%

%

2

%

3

4

%

5

%

6

%

205

210

215

220

225

230

235

240

245

250

255

5

7

8

9

10

11

12

13

14

15

6

指令延迟

)

ps

(

性能改善

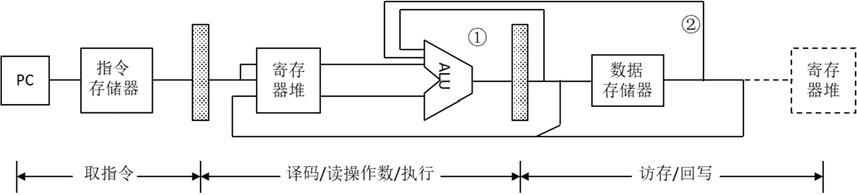
)

%

(

3.

1. 会。beq 的比较判断电路只能前移至译码/读操作数/执行阶段。这与本章介绍的处理是完全一致的。可能被清除的指令仍然是 beq 后面的那条指令，因此只有 1 条。
2. 从 ALU 和 DM 分别向 ALU 的 A 端转发数据。注意，由于 RF 无内部转发，因此必须有路径 2。



1. 可以。因为对于{lw，add}来说，rs 只在 ALU 的 A 端使用。无论当前哪条指令需要使用 ALU，其前面只可能有 1 条指令位于最后一级。而无论是哪条指令位于最后一级，结果均已产生，因此必然可以通过旁路转发来消除数据冒险。

4.

1. 在 3 级流水线中，中间那段的延迟已经是最坏的 2L 了。
2. 转发电路的组合逻辑为 DM 读出+ALU，同样也是 2L。
3. 这表明转发并没有增加最坏延迟，因此不会导致性能下降。

5.

1. 指令 1 和指令 2 之间在$1 有数据冒险，在转发电路存在的前提下，仍然必须暂停

1 个时钟周期。指令序列调整为：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | lw $1, 0($2) nop  addi $1, $1, $1 sw $1, 0($2) lw $1, 4($2) sw $1, 8($2) |

原指令 4（现指令 5）与原指令 5（现指令 6）之间存在数据冒险。但原设计没有转发电路，因此必须将 sw 冻结在 IF/ID，直至 lw 进入 MEM/WB。此时通过 RF 内部转发，sw 就能得到正确的$1。暂停周期数为 2。 故上述代码总的暂停时间为 3 个时钟周期。

1. 前问的 2 个暂停周期是因为解决 lw-sw 之间的数据冒险而产生的，因此需要从

MEM/WB 增加一个转发至 DM 的 WD。

DM



WD

EX/MEM

MEM/WB

0

1

6.

1. load-R：没有数据数据相关时，load 的 CPI 为 1。如果有数据相关，需暂停 1 个时钟周期，其 CPI 为 2。
2. load-store：没有数据数据相关时，load 的 CPI 为 1。如果有数据相关，需暂停 2 个时钟周期，其 CPI 为 3。
3. store：题目中未出现数据相关，因此 CPIstore 为 1。
4. 分支：预测成功，分支的 CPI 为 1。如果预测失败，需暂停 1 个时钟周期，分支的CPI 为 2。
5. R 型：CPI 为 1。

1) lw-sw 在$1 有 2 次相关；lw-sub 在$1 相关；add-sub 在$3 相关。

2)

I1: lw $1, 0($2)

I3: add $3, $2, $2

I5: or $5, $5, $6 I2: sw $1, 0($1)

I4: sub $4, $1, $3

3) 优化前：13 个 cycle

由于仅在 W 和 D 之间存在转发，因此 lw-sw 以及 add~sub 之间必须分别插入 2 个 NOP。指令总数从5条变为9条，因此流水线共计需要9+（5-1）=13个时钟周期。 优化后：9 个 cycle

优化后，lw~sw 和 add~sub 的数据相关均通过转发解决了，因此无需插入 NOP，故执行时间=5+（5-1）=9 个时钟周期