4.3. 流水线的数据通路与控制

流水线（pipelining）是一种实现多条指令重叠执行的技术，与生产流水线类似。

在单周期实现中将指令的执行划分为 5 个阶段（IF, ID, EX, MEM, WB），意味着采用 5 级流水线，也就意味着在任何一个时钟周期内，最多会有 5 条指令在执行。

4.3.1. 流水线数据通路

在流水线的某一级前，需要用一个寄存器来保存来自之前级的数据通路部件的数据，以便将之前级的数据通路部件腾出供下一条指令使用，同时可以保证本级的正常运行，称为流水线寄存器（pipeline registers）。

值得注意的是，所有指令都会更新数据通路中的某些状态（如寄存器堆、存储器和 PC 等），因此完整指令执行后无需流水线寄存器保存被本指令更新过的状态单元的数据，因为如果需要这些数据，下一条指令直接从相应的状态单元中取即可。

因此，流水线的数据通路额外有 4 个流水线寄存器：IF/ID（ID 前），ID/EX（EX 前），EX/MEM（MEM 前），MEM/WB（WB 前）。PC 寄存器可以看做一个流水线寄存器，给 IF 级提供数据。与以上 4 个流水线寄存器不同，PC 寄存器在发生异常时必须保存它的内容，其他流水线寄存器则可以被丢弃。

不包含跳转指令的数据通路的流水线版本如 Figure 4-32 所示：

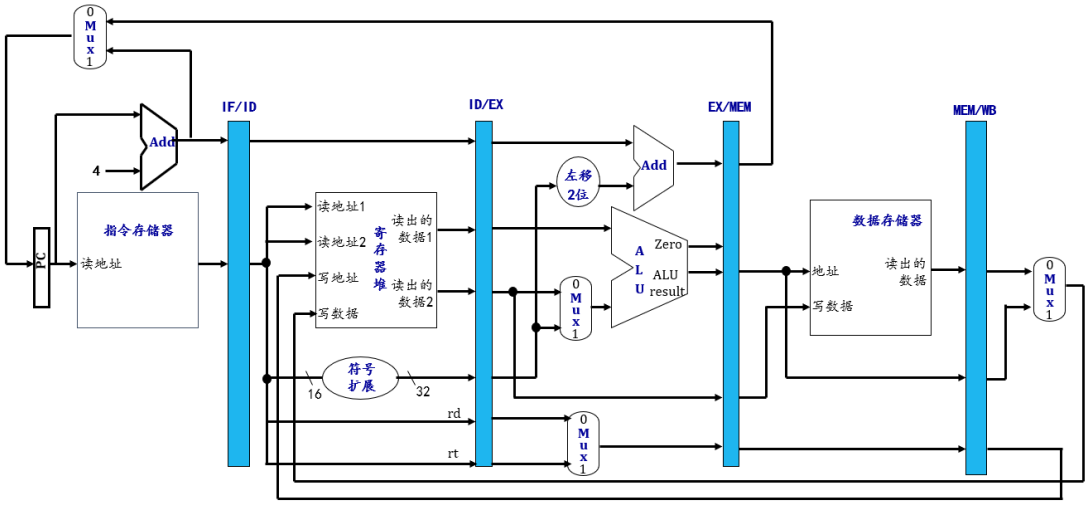

Figure 4-32 The Pipelined Version of the Datapath

有两种基本的流水线图形化表示方法：单时钟周期流水线图和多时钟周期流水线图。我们约定，当寄存器或存储器被读取时，其右半部分突出显示；当它们被写入时，其左半部分突出显示。

单时钟周期流水线图

单周期流水线图一般用于展示一条指令的执行细节，这里以 lw 指令为例，其在流水线各级的执行过程如 Figure 4-33 至 Figure 4-37 所示：

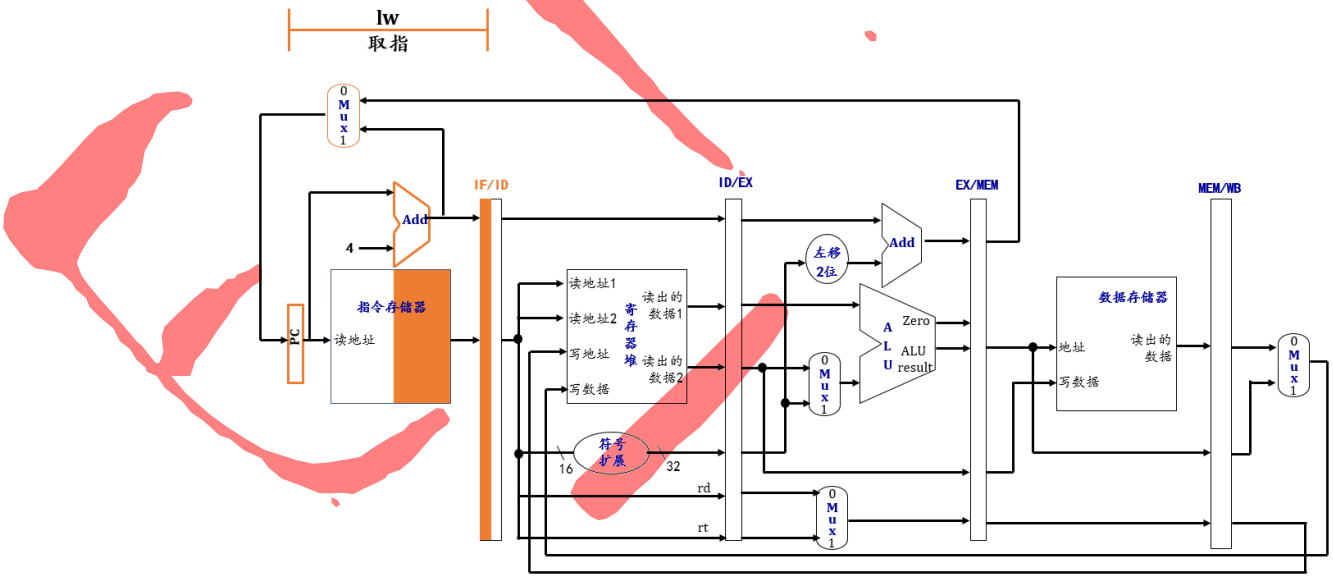

Figure 4-33 IF: The 1st Pipe Stage of lw
Read PC and PC+4

取指阶段：将 PC+4（32 位）和所取得的指令（32 位）放入 64 位的 IF/ID 流水线寄存器。

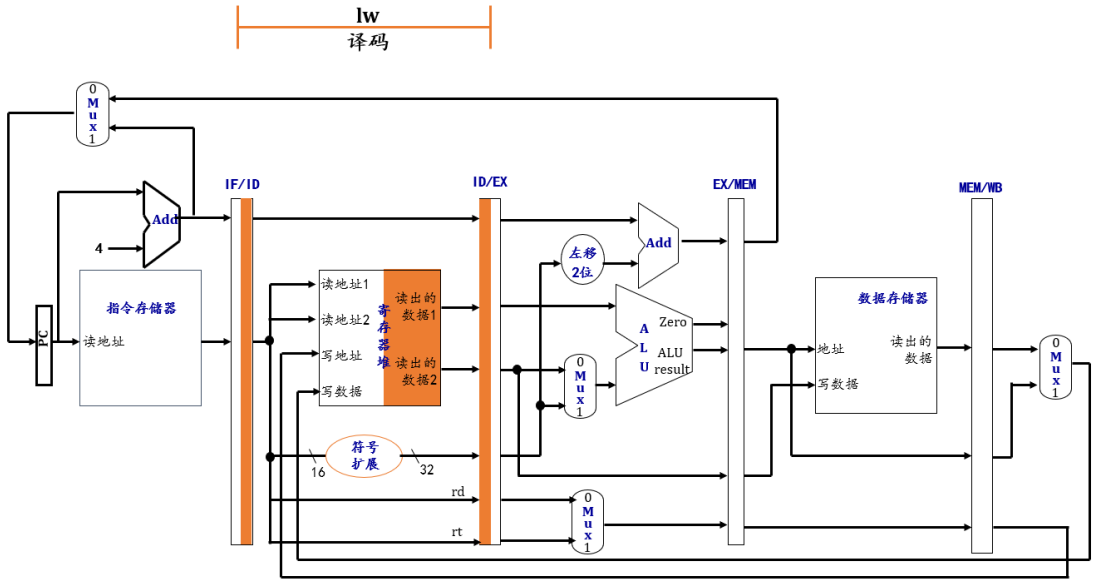

Figure 4-34 ID: The 2nd Pipe Stage of lw
Decode Instruction, read Registers, decide rt and sign-extend

译码阶段：将 PC+4（32 位），读出的数据 1（32 位），读出的数据 2（32 位），rt 和 rd 寄存器号（10 位，在 EX 级根据不同指令进行选择）和经过符号扩展后的 32 位立即数放入 138 位的 ID/EX 流水线寄存器。

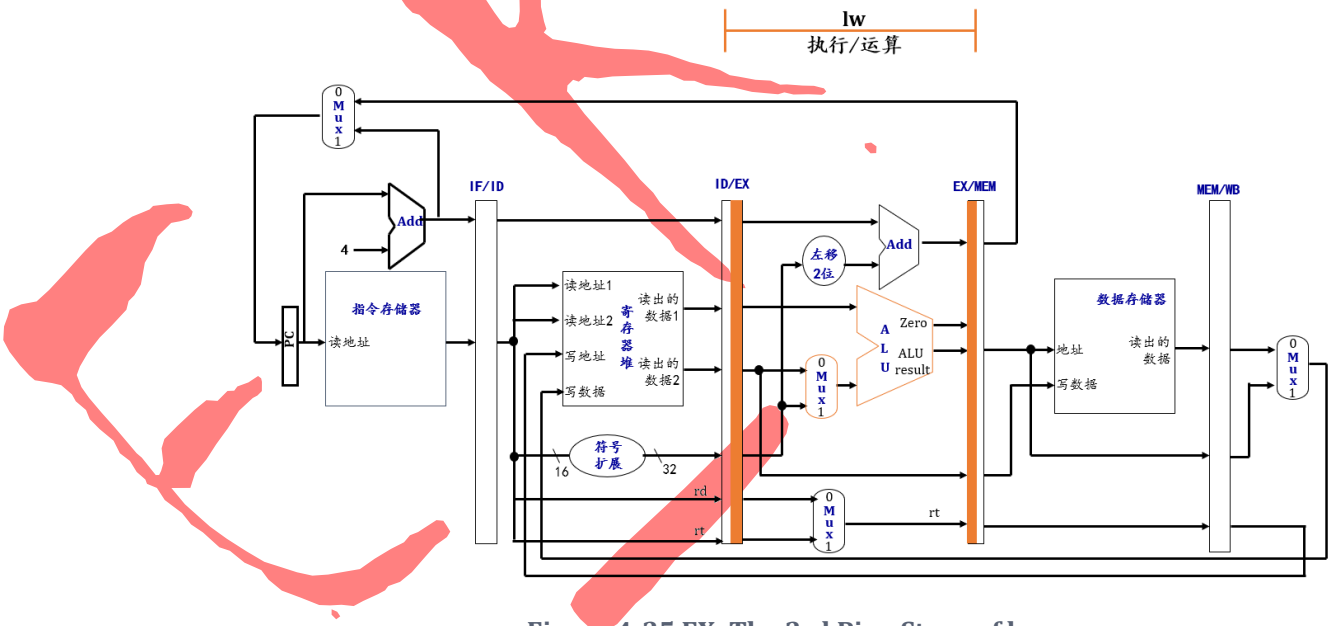

Figure 4-35 EX: The 3rd Pipe Stage of lw

ALU compute

Place ALU result, ALU zero, Add result, rt number and Read data 2 in EX of 102 bits

执行/运算阶段：将 ALU 计算结果（32 位），ALU zero（1 位），读出的数据 2（32 位），rt 寄存器号（5 位）和 Add 计算结果（32 位）102 位的 EX/MEM 流水线寄存器。

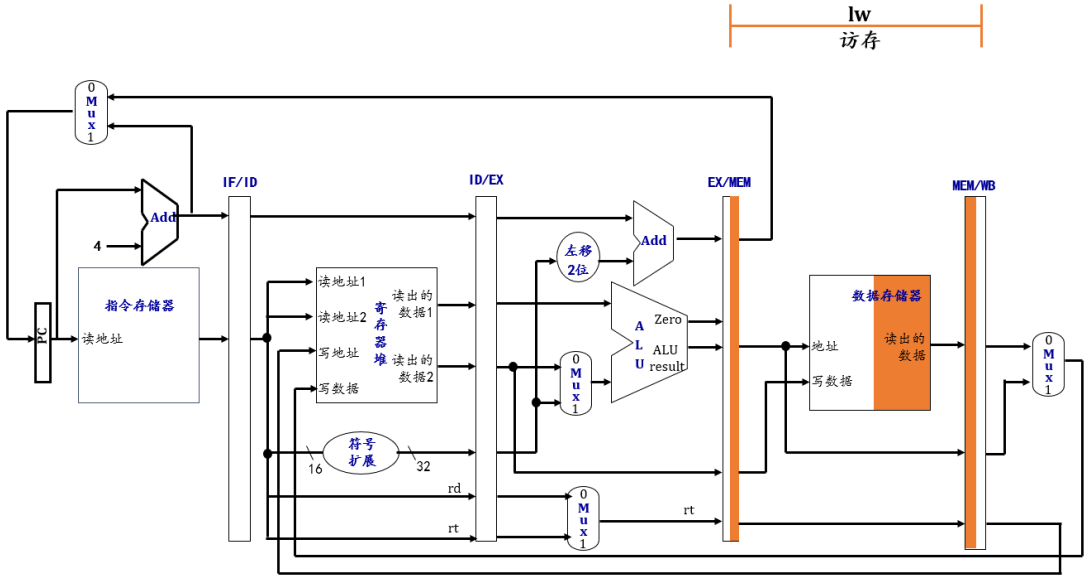

Figure 4-36 MEM: The 4th Pipe Stage of lw

Read D-Mem

访存阶段：将从内存读取的数据（32 位），ALU 计算结果（32 位）和 rt 寄存器号（5 位）69 位的 MEM/WB 流水线寄存器。

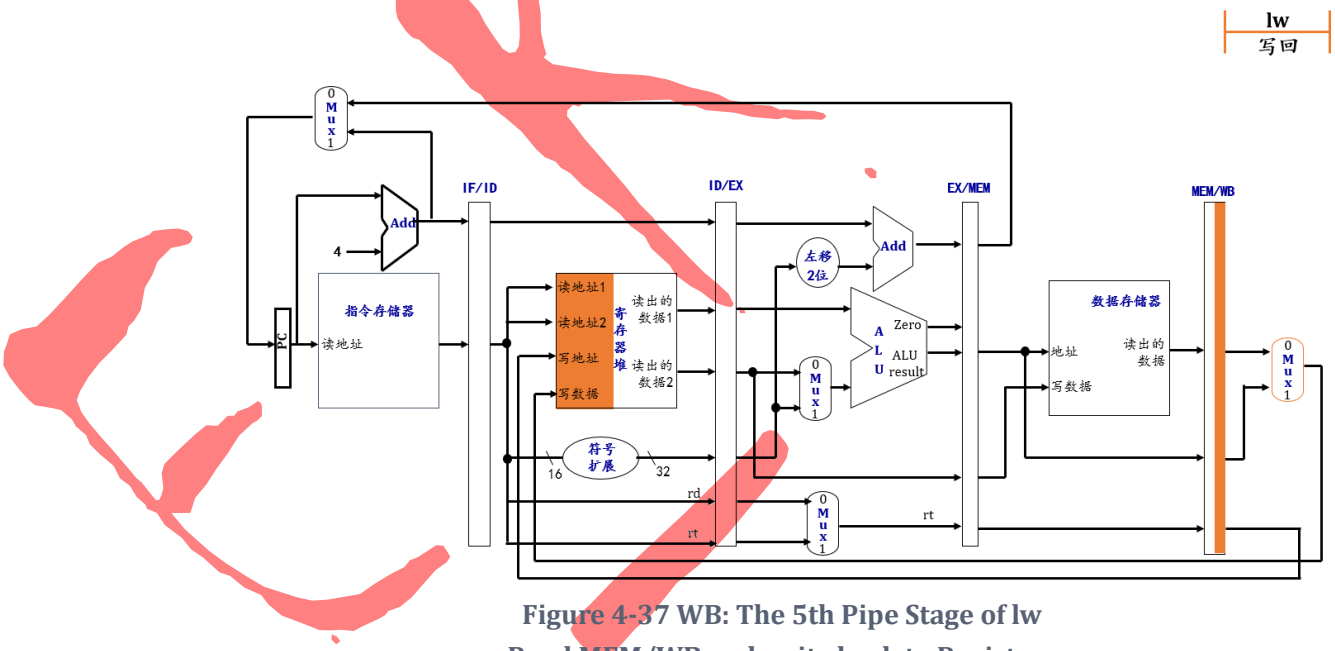

Figure 4-37 WB: The 5th Pipe Stage of lw
Read MEM/WB and write back to Registers

多时钟周期流水线图

多周期流水线图用于对流水线总体进行全局描述，可以展示一个指令序列的执行情况，省略每条指令的单周期实现细节。我们约定写操作发生在时钟周期的前半段，读操作发生在时钟周期的后半段。以下列 5 条指令序列为例：

```
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
```

Figure 4-38 和 Figure 4-39 展示了以上指令序列的两种多时钟周期流水线图：

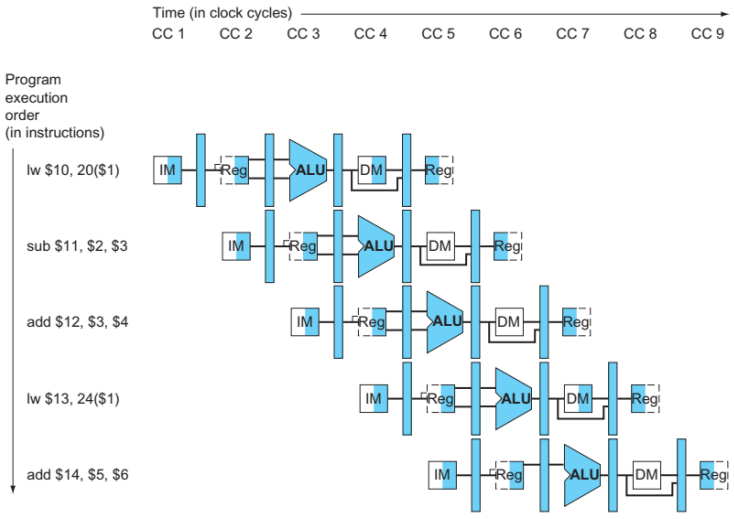

Figure 4-38 Multiple-Clock-Cycle Pipeline Diagram of the Five Instructions Above

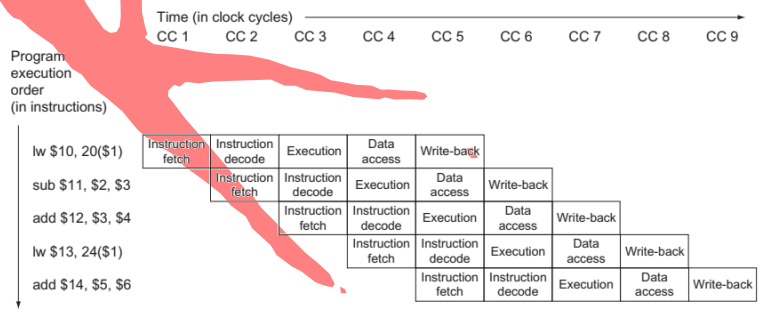

Figure 4-39 Traditional Multiple-Clock-Cycle Pipeline Diagram of in Figure 4-38

Note

虽然流水线有单周期和多周期两种图示，但其与多周期实现有着本质区别。多周期实现是指一条指令在不同的时钟周期可以多次使用同一个功能部件，即功能部件的共享可以在一条指令中实现；而流水线是多条指令重叠执行，即功能部件的共享只能在不同指令中实现。

流水线是在单周期实现的基础上设计的，单条指令除个别特殊操作外，只能从左到右向前推进，不能使用前面级的部件。

4.3.2. 流水线控制

实现流水线控制就是为每一条指令的每一个执行级将 9 个控制信号设置为合适的值。与 PC 一样，4 个流水线寄存器在每一级都会被写入，所以也不需要单独的写控制信号。最简单的实现方法就是扩展流水线寄存器，使之包含这 9 个控制信号。

控制信号在 ID 级产生，从 EX 级开始。EX 级使用到 ALUOp，ALUSrc，RegDst 和 Jump 共 4 个控制信号，MEM 级使用到 MemRead，MemWrite 和 PCSrc 共 3 个控制信号，WB 级使用到 RegWrite 和 MemToReg 共 2 个控制信号。同时，上一级的流水线寄存器需要保存之后级使用到的控制信号，如 Figure 4-40 所示：

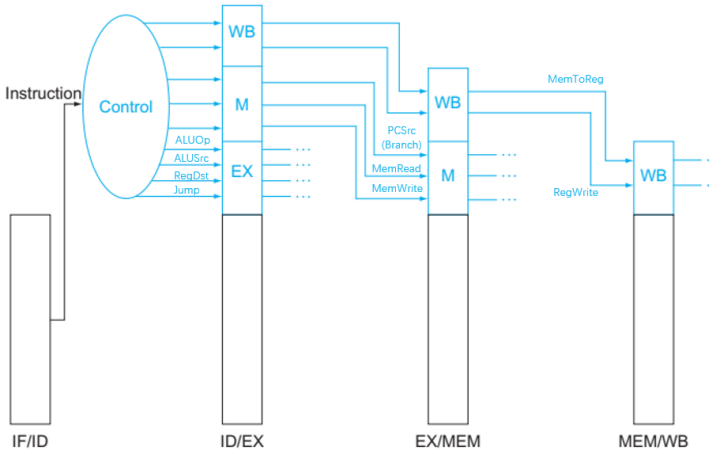

Figure 4-40 The Control Lines for the Final Three Stages

添加了控制信号的流水线数据通路最终如 Figure 4-41 所示：

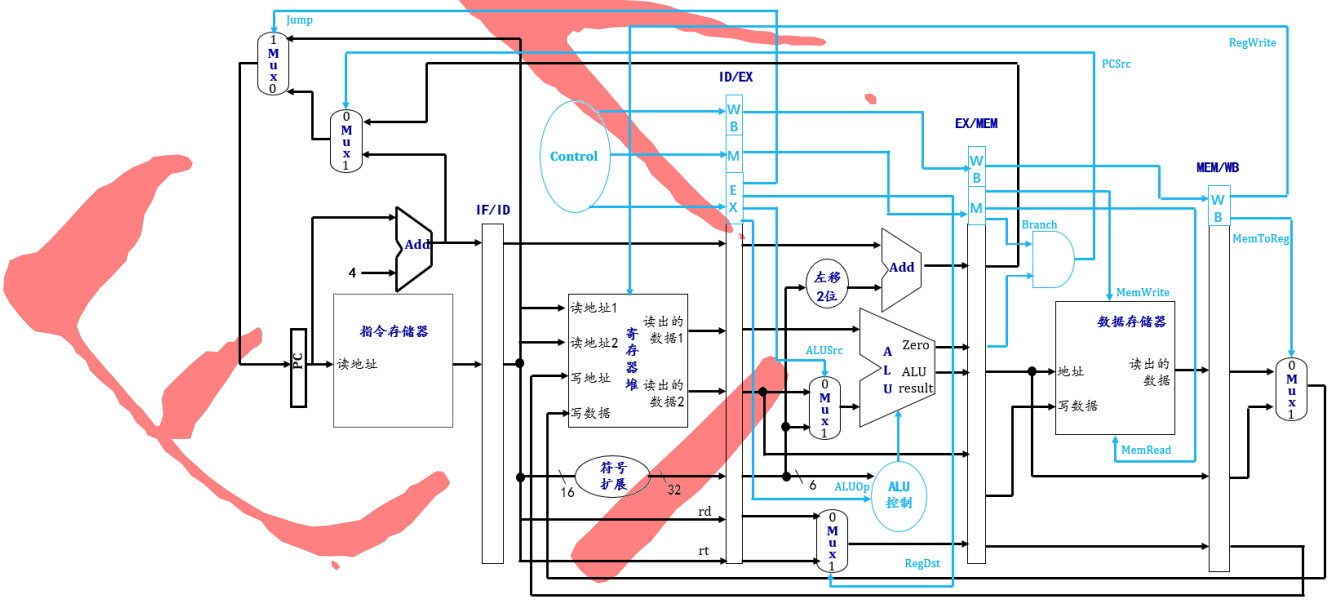

Figure 4-41 The Pipelined Datapath with the Control Signals

4.3.3. 流水线性能

本章基于单周期实现的流水线设计将一条指令分 5 级进行，将流水线的一级看成一个时钟周期，即指令在多个周期内完成。一般时钟周期是固定的，所以时钟周期必须满足最慢操作级的执行时间。

在本节，假设流水线能够顺畅进行，各指令间不相互依赖。

假设各流水级的操作平衡（执行时间相同），且流水线不引入额外开销，则这种理想情况下的流水线处理器中的指令间执行时间为：

$$\text{Time between instructions}_{pipelined} = \frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of pipe stages}} \qquad 4.1$$

Example 4.3

假设一个 5 级流水线处理器的各级执行时间为：IF（200ps），ID（100ps），EX（200ps），MEM（200ps），WB（100ps）。

(1) 分别求出单周期非流水线实现的时钟周期和流水线实现的时钟周期；

(2) 以上两种方式处理完成 1 000 003 条指令的执行时间是多少？

(3) 求出流水线实现相对于非流水线的相邻指令执行时间比以及处理完成 7 条和 1 000 003 条指令的加速比。

(1)

非流水线：200+100+200+200+100=800ps

流水线：max(200, 100, 200, 200, 100)=200ps

(2)

非流水线：1 000 003×800=800 002 400ps，如 Figure 4-42 所示

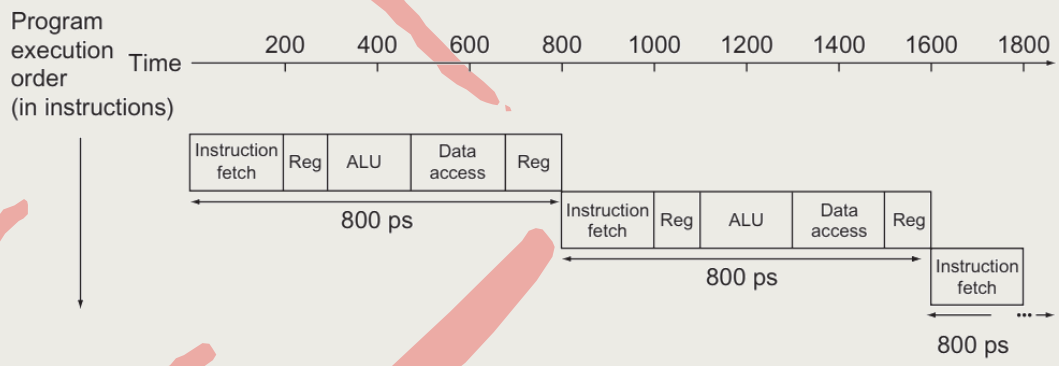

Figure 4-42 Single-Cycle, Nonpipelined Execution

流水线：2×(5-1)×200+1 000 003×200=200 002 200ps，如 Figure 4-43 所示

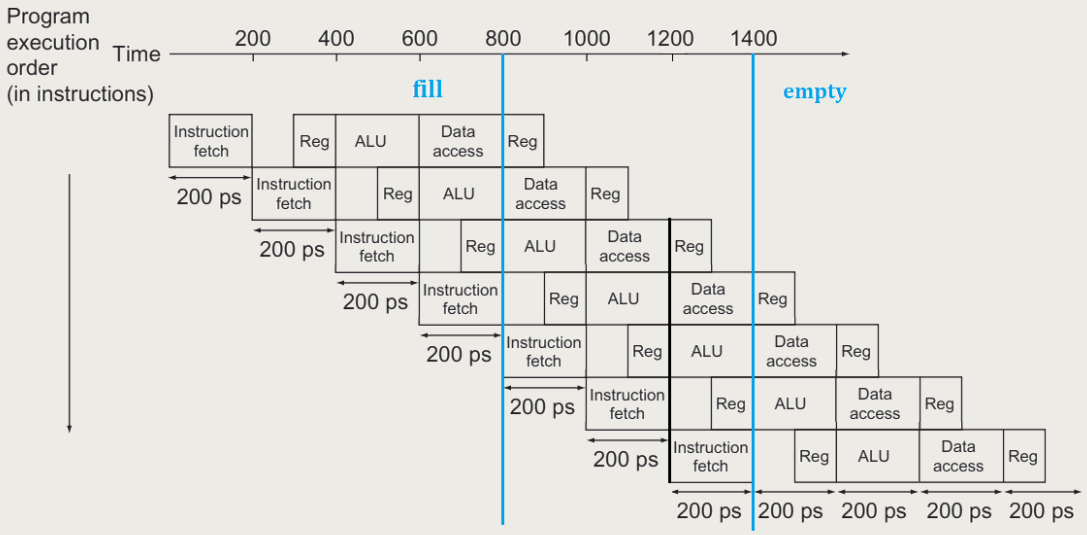

Figure 4-43 Single-Cycle, Pipelined Execution

(3)

相邻指令间隔时间比：800/200=4

7 条指令执行完成的加速比： 5600/2000=2.8

1 000 003 条指令执行完成的加速比：800 002 400/200 002 200=3.99997≈4

Note

第(1)问可以看出，实际中流水线各级操作并不平衡，所以其时钟周期并不是理想情况下的 800/5=160ps。

如 Figure 4-43 所示，流水线有一个填满和清空的过程，分别消耗（级数-1）×时钟周期的时间。在这两个过程之间，每经历一个时钟周期就完成一条指令。流水线的加速比永远不能达到理想情况除了实际实现的原因，还有流水线本身运行的原因。即流水线想要运行起来，必须经历填满和清除的过程，这也使得加速比只能无限趋近于相邻指令间隔时间比。

流水线所带来的性能提高是通过增加指令的吞吐量（在给定的时间内完成更多的指令），而不是减少单条指令执行时间来实现的，甚至可能因为额外开销而增加单条指令的执行时间。

当指令数量足够多时（实际上程序都会执行成千上万条指令），流水线的优势便体现出来。此时我们近似（除去头尾填充和清除阶段的指令）看作每个时钟周期完成一条指令，虽然实际上是完成五条指令的各 1 级，将该时钟周期执行取指操作的指令视为本周期执行的指令。