第1章 MIPS 五段流水 CPU 设计实验

1.1 MIPS CPU 上板实验

1.1.1 实验目的

学生掌握 N4-DDR FPGA 开发板的使用，能将课程实验中设计完成的单周期 CPU 在该平台上具体实现，并能正确运行标准测试程序。

1.1.2 实验内容

在课程实验中已经利用运算器实验，存储系统实验中构建的运算器、寄存器文件、存储系统等部件在 Logisim 平台中构建了一个 32 位 MIPS CPU 单周期处理器，该处理器应支持 20 余条基础指令，另外还支持扩展指令集中的 2 条 C 类运算指令，1 条 M 类存储指令，1 条 B 类分支指令。采用团队合作的形式将 Logisim 平台中设计单周期 CPU 移植到 N4-DDR FPGA 开发板进行具体实现，最终能在开发板上正确运行标准测试程序以及差异化指令集测试程序。

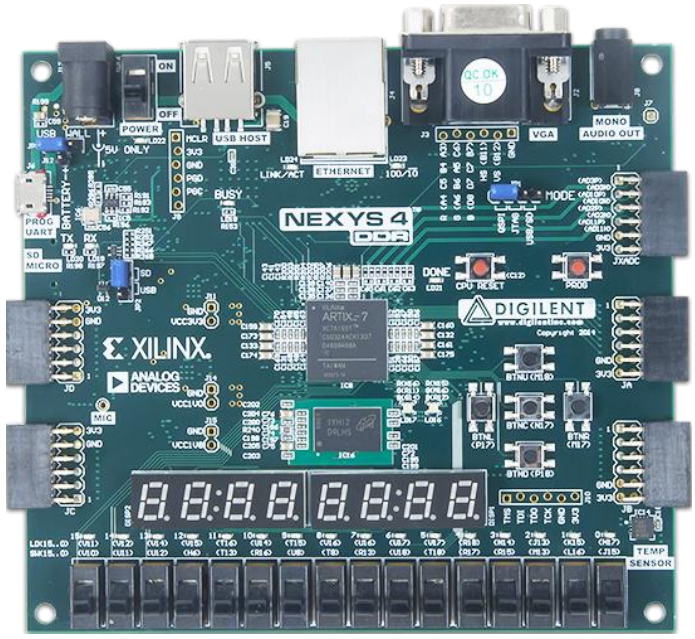

图 1.1 NEXYS 4 开发板

实验要求：

(1) 能在 FPGA 开发板上利用 8 个 8 段数码管进行正确的显示输出，另外可以通过拨码开关实现显示功能的切换，如正常数据输出显示、不同内存地址内存值的观察、PC 值观察、时

钟周期数观察、运行参数统计观察（需要设置显示功能切换拨码开关---用于切换显示功能，需要设置地址输入拨码开关）。

(2) 支持频率切换，最高频率以及演示频率（保证输出显示效果与 Logisim 相同，肉眼可观察），需要设置频率切换拨码开关。

(3) 支持总复位按钮，按下总复位按钮后，系统复位，所有寄存器，存储器值清零，从 0 号地址开始重新执行程序。

1.1.3 实验步骤

(1) **选择最优方案**，本实验采用团队合作形式完成，代码可共享，但必须支持各自的差异化指令，课程实验时所有同学已经采用 Logisim 实现了自己的单周期 CPU，并实现了差异化指令，小组同学间方案可能略有差异，因此第一步需要团队集体决策，选择一个结构最为合理的 Logisim CPU 版本作为 FPGA CPU 设计的参照模板，选择一个好的参考方案将为后续工作带来很大的便利，如果无法达成一致，也可以共享功能部件，数据通路以及控制器部分由各小组成员自行开发。

(2) **功能部件实现**，将 Logisim 平台课程实验中设计完成的功能部件逐一利用硬件描述语言实现，基本功能部件包括：运算器 ALU、指令存储器、程序计数器 PC、寄存器文件、数据存储器，数据位扩展器，多路选择器、运行参数统计模块、分频模块、数码管扫描显示模块、LED 指示模块和显示开关切换模块。除 ALU 建议安排 1 名同学完成外，其它都可以考虑 1 名同学负责 2-3 个模块，例如：指令存储器和数据存储器基本上可以通过一个模块统一实现；也可以同时安排 2 名同学来完成相同的任务，这样的好处是可以商量；每个模块 Verilog 程序编写完成后同样要通过仿真确保模块的正确性，不进行模块功能仿真将为后续整体联调带来极大的困难。

(3) 控制器设计，单周期 MIPS CPU 的控制器无状态机机，直接演变成一个纯组合逻辑电路；用硬件描述语言实现时构建一个模块就可以了，建议安排 2 名同学来做，由于 Logisim 是用表达式生成的，可以利用硬件描述语言连续赋值语句直接转化，控制器实现的工作量主要是该模块的具体调试，一定要通过仿真保证每条指令控制信号输出的正确性。

(4) **数据通路连接**，将实现的运算器 ALU、指令存储器、程序计数器 PC、寄存器文件、数据存储器，数据位扩展器，多路选择器、运行参数统计模块、分频模块、数码管扫描显示模块、LED 指示模块和显示开关切换、控制器等模块按照 Logisim 图用结构化描述方法连接起来构成数据通路，同时根据需要添加一些其它必要的电路，例如与门、非门、锁存器等；数据通路连接完成后一定要仔细检查连接的正确性，特别是连线（wire）的位宽要匹配；

(5) **功能仿真**，在指令存储器中固化测试程序，通过功能仿真验证单周期 CPU 实现的正确性，功能仿真结束后还需进行时序仿真确保能正确上板运行。

(6) **实际测试**,按照 Nexys4 DDR FPGA 开发板使用手册中的规范绑定引脚，生成 Bit 流；为了演示的方便，建议 Bit 流要生成 2 个版本的，一个是正常时钟频率的，这个版本时钟频率最好能够达到 100MHz；另一个是降低时钟频率的版本，使之能够演示出 Logisim 上一样的运行效果。

(7) **差异化实现**,在小组公共版本基础上实现自己的 CCMB 指令，并进行测试，完成后提交检查。

1.2 理想流水线 CPU 设计实验

1.2.1 实验目的

学生了解 MIPS 5 段流水线分段的基本概念，能设计流水接口部件，将课程实验中设计完成的单周期 CPU 改造成理想流水 CPU，并能正确运行无冒险冲突的理想流水线标准测试程序，能根据时空图简单分析流水 CPU 性能。

1.2.2 背景知识

MIPS 单周期 CPU 设计实现简单，控制器部分是简单的组合逻辑电路，但该 CPU 所有指令执行时间均是一个相同的周期，即以速度最慢的指令作为设计其时钟周期的依据，如图 1.2 所示，单周期 CPU 的时钟频率取决于数据通路中的关键路径（最长路径），所以单周期 CPU 设计效率较低，性能不佳，现代处理器中已不再采用单周期方式设计处理器，取而代之的多周期设计方式，而多周期 CPU 中流水 CPU 设计是目前的主流技术。

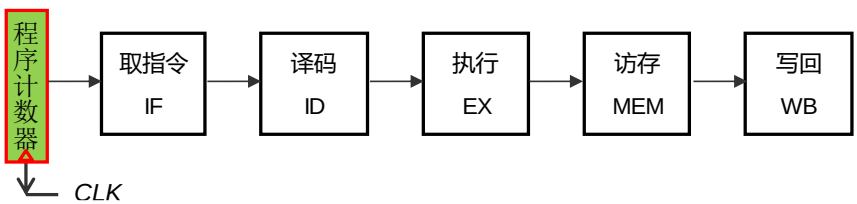

图 1.2 单周期 CPU 逻辑架构

流水线处理技术并不是计算机领域所特有的技术。在计算机出现之前，流水线技术已经在工业领域得到广泛应用，如汽车装配生产流水线等。计算机中的流水线技术是把一个复杂的任务分解为若干个阶段，每个阶段与其它阶段并行运行。由于其运行方式和工业领域中的流水线处理技术十分类似，因此被称为流水线技术。

把流水线技术应用于运算的执行过程，就形成了运算操作流水线，如浮点数加法运算过程可分解为求阶差、对阶、尾数加和规格化 4 个阶段。把流水线技术应用于指令的解释执行过程，就形成了指令流水线，如图 1.3 所示，MIPS 指令流水线通常将指令执行过程分为取指令 IF、指令译码 ID、指令执行 EX、访存 MEM、写回 WB 共五个阶段，在每个阶段的后面都需要增加一个锁存器（又称为流水接口部件，用于锁存上一阶段的加工数据或处理结果），以保证该阶段的执行结果给下一个阶段使用。

程序计数器、所有锁存器均采用公共时钟进行同步，每来一个时钟，各阶段功能部件逻辑新处理的数据将锁存到后段的锁存器中。由于 5 个阶段逻辑延迟时间并不一致，为保证指令流水线正确运行，最大时钟频率取决于 5 段中最慢一段的关键路径，所以分段时应该尽量让各阶段时间延迟相等，假设各阶段时间延迟均为 T。锁存器在公共时钟的驱动下可以锁存流水线前

段逻辑加工完成的数据，以及相应的控制信号，锁存的数据和信号将用于后段的继续数据加工或处理。

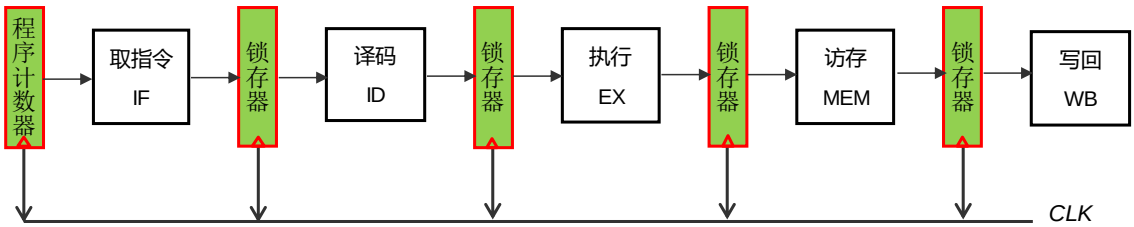

图 1.3 指令流水线逻辑架构

单周期 CPU 中由于各段组合逻辑完全串联，无法并行运行，其时空图如图 1.4 所示，图中，横坐标表示时间，也就是连续指令进入到流水线到输出流水线所经过的时间，当流水线中各阶段延迟时间相等时，横坐标被分割成相等长度的时间段，这里一个时间段为 T；纵坐标表示空间，即流水线的各阶段对应的功能部件。单周期 CPU 执行一条指令的时间为 5T。

MIPS 5 段流水 CPU 中由于引入了锁存器，所以各段可以并行运行，如图 1.5 所示，当流水线充满后，每隔一个时钟周期 T，系统将流出一条执行完毕的指令，相比单周期 CPU 提升 5 倍。如需要执行 n 条指令，执行时间为(n+4)*T，当然这是理想情况，实际指令流水线存在很多冒险冲突，本实验实现的是无冒险冲突的理想流水线，所以暂时可以不考虑这些冒险冲突。

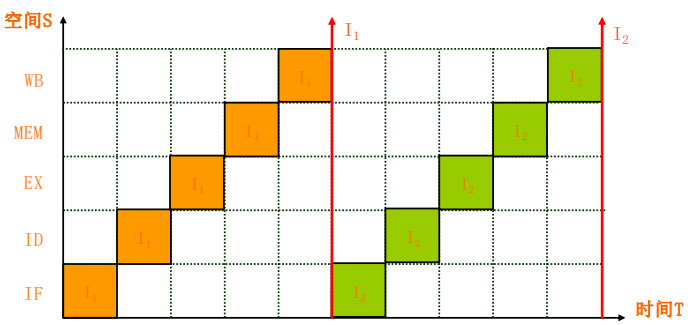

图 1.4 单周期 MIPS CPU 时空图

图 1.5 五段流水 MIPS CPU 时空图（无冲突）

1.2.3 实验内容

将课程实验完成的 MIPS 单周期 CPU 改造成支持无冒险冲突程序运行的理想流水线架构，标准测试程序如下，该程序在数据存储器 0,4,8,12 位置依次写入数据 0,1,2,3，程序无任何数据冒险和分支冒险，所以设计流水线时无需考虑任何冒险冲突的处理。

```
//file: 理想流水线测试.asm
#理想流水线测试，所有指令均无相关性，一共17条指令，
#5段流水线执行周期数应该是5+(17-1）=21
  addi $s0,$zero, 0
  addi $s1,$zero, 0
  addi $s2,$zero, 0
  addi $s3,$zero, 0
  ori $s0,$s0, 0
  ori $s1,$s1, 1
  ori $s2,$s2, 2
  ori $s3,$s3, 3
  sw $s0, 0($s0)
  sw $s1, 4($s0)
  sw $s2, 8($s0)
  sw $s3, 12($s0)
  addi $v0,$zero,10          # system call for exit
  addi $s1,$zero, 0          #三条无用指令消除syscall与v0寄存器的相关性
  addi $s2,$zero, 0
  addi $s3,$zero, 0
  syscall                    # 停机
```

单周期 MIPS CPU 架构如图 1.6 所示，该架构将在一个时钟周期内完成取指令，指令译码，运算，访存，写回等一系列动作，各阶段直接串行连接，时钟周期等于各阶段的总时延迟，性能较差。要将单周期架构改造成流水线架构，首先需要将将 MIPS 指令执行过程严格分成 5 个阶段：取指令 IF 段、指令译码 ID 段、指令执行 EX 段、访存 MEM 段、写回 WB 段，(注意不得简化成 4 段流水线)。其中 IF 段包括程序计数器 PC，NPC 下址逻辑，指令存储器等功能模块；指令译码 ID 段包括控制器逻辑、取操作数逻辑；指令执行 EX 段主要包括运算器模块；访存 MEM 段主要包括内存读写模块，写回 WB 段主要包括对寄存器写入控制模块。需要注意的是不是所有指令都需要经历完整的 5 个阶段。

不同阶段之间增加流水接口部件（锁存器），如图 1.7 所示，在单周期 CPU 实现基础上需要增加 IF/ID、ID/EX、EX/MEM、MEM/WB 共 4 个流水接口部件，4 个流水接口均采用公共时钟进行同步，流水接口定义尽可能简化，在 Logisim 中实现时其内部主要是若干寄存器，用于锁存段间数据，五段流水结构在数据表示实验中已经出现过，具体实现时可以参考数据表示实验

中海明编码流水传输电路。

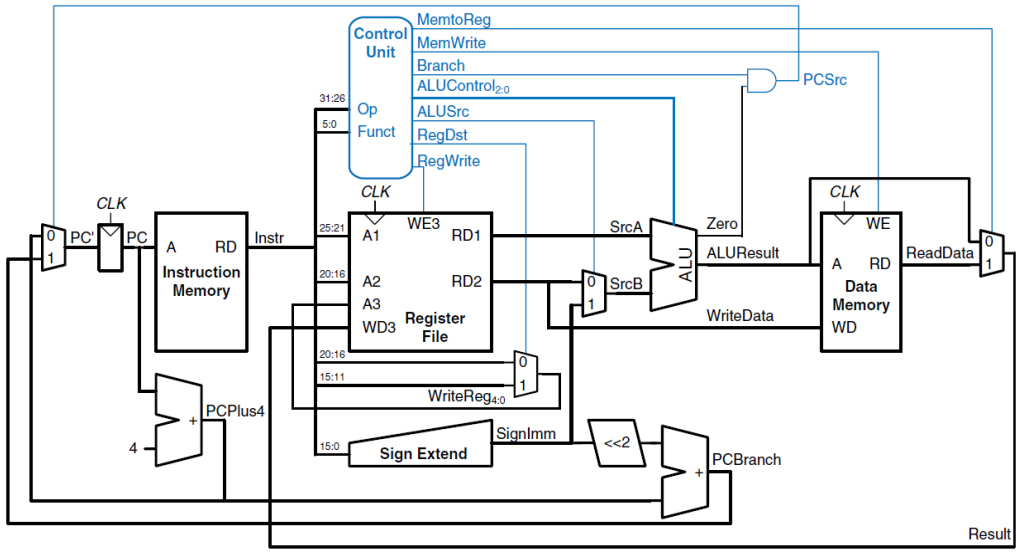

图 1.6 单周期 MIPS CPU

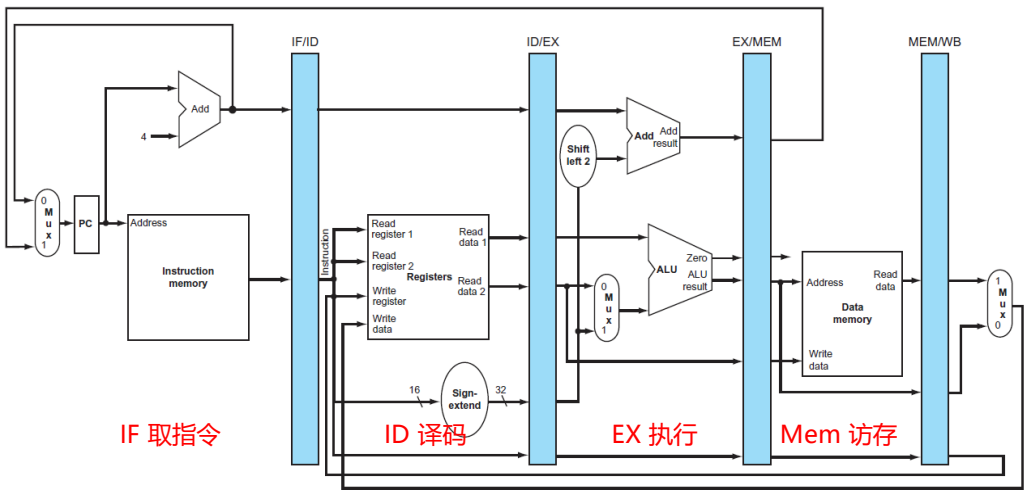

图 1.7 五段流水 MIPS CPU

流水线通过流水接口部件为后续段提供数据信息，控制信息，如图 1.8 所示，向前段传递反馈信息，流水线后段部件对数据的加工处理依赖于流水线前段通过流水接口部件传递过来的信息。ID 段译码生成该指令的所有控制信号（中蓝色信号为控制信号），控制信号将通过锁存器逐段向后传递，后段功能部件所需的控制信号不需要单独生成，直接从锁存器获取即可。注意单周期 CPU 中的控制器可以在译码 ID 段直接复用。不同的流水接口部件锁存的数据和控制信号不同，具体可根据前后阶段之间的交互信息进行考虑，以最为复杂的 ID/EX 接口部件为例，该锁存器锁存译码阶段由控制器产生的所有控制信号，同时还需要锁存由取操作数部件取出的寄存器的值或者立即数的值，ID/EX 部件设计完成后，其他各段流水接口部件可以直接复

制后进行适当精简。

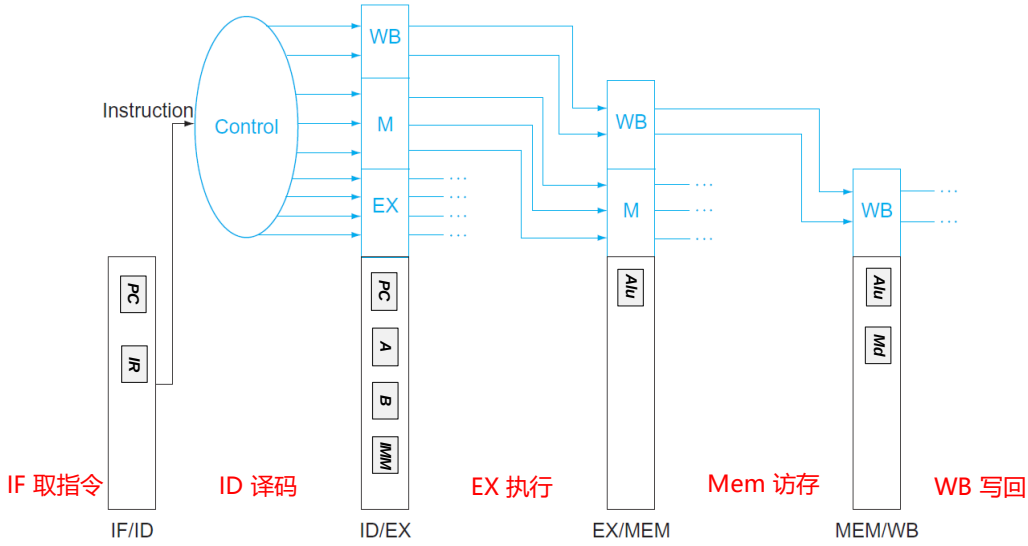

图 1.8 五段流水 MIPS CPU 数据与控制信号传递

注意事项：

(1) 在 Logisim 平台实现时应尽量缩小原单周期 CPU 原理图中的功能部件尺寸，以方便布局绘图，如图 1.9 所示，寄存器文件，运算器等功能部件都重新进行了封装，方便布局。

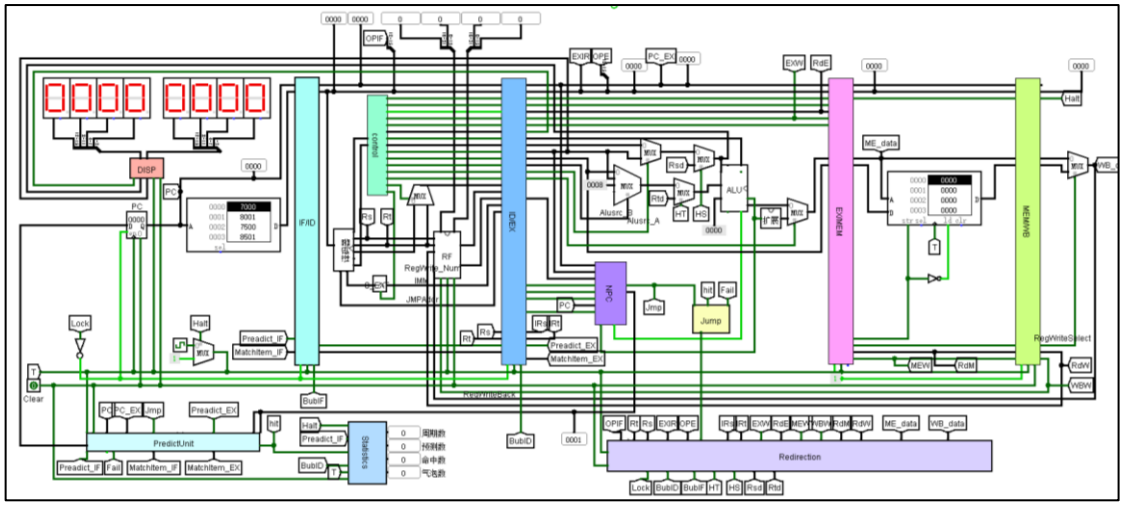

图 1.9 五段流水 MIPS CPU 原理图布局

(2) 流水接口部件封装尽可能封装的长一点，否则连线过多会给布线带来困扰。通过流水接口部件向后传输的控制信号应遵循越晚用到越靠近顶端的原则，便于腾出更多的空间进行流水功能段的电路布局。适当使用颜色标记关键功能部件和流水接口部件。

(3) 指令存储器 ROM 和数据存储器 RAM 必须在 main 电路中可见，显示模块应该在主电路中可见。不能封装在子电路中，以便于观察和调试。
(4) 主要数据通路应直接连接，横向可连接的线缆尽量直接连接，避免隧道的滥用，以保证原理图的可读性,连线一般不允许穿越其他功能部件的封装。
(5) 尽可能使用标签工具注释电路，包括控制信号，数据通路，显示模块，总线等，会使得电路更加容易调试。注意标签以及注释的命名规范,过长的命名会影响原理图布局。
(6) 可以使用任何 logisim 内建的任何电路组件构建原理图。
(7) **严禁对时钟信号进行逻辑门操作，**停机操作也可以通过相关功能部件使能端完成。

1.2.4 实验步骤

(1) **设计流水接口部件**，首先考虑 4 个流水接口部件需要锁存的具体信息，然后首先设计最为复杂的 ID/EX 流水接口部件,封装设计完成后再复制生成 IF/ID、EX/MEM、MEM/WB 流水接口部件。注意流水接口部件封装的长度、标签注释以及颜色区分。
(2) **重新封装较大尺寸的功能部件**，如运算器，寄存器文件等。
(3) **重构流水 CPU**，将单周期 CPU 中的数据通路全部删除，重新布局原理图，将各段功能部件与流水接口部件进行连接。
(4) **功能调试**，加载标准测试程序进行功能调试。

1.3 气泡流水线实验

1.3.1 实验目的

学生理解数据相关的基本原理，掌握利用插入气泡方式消除指令数据相关性的方法，能够在理想流水线 CPU 的基础上增加逻辑电路检测数据相关性，可通过硬件插入气泡的方式消除数据相关性；学生理解控制相关的基本原理，理解其引起的流水线停顿导致的性能下降，掌握分支相关流水线处理机制，并能够增加相关逻辑使得流水线能正确处理分支指令引起的控制相关；学生理解结构相关的基本原理，能分析五段流水 MIPS 中存在的结构相关问题，并能运用适当的方案进行解决,最终实现的 CPU 能正确运行单周期 CPU 实验中测试过的 benchmark 标准测试程序。

1.3.2 背景知识

理想的流水线所有待加工对象均需要通过同样的部件(阶段),不同阶段之间无共享资源，且各阶段传输延迟一致，进入流水线的对象也不应受其他阶段的影响，但这仅仅适合工业生产流水线，计算机指令流水线存在较多的指令相关，会引起流水线的冲突和停顿。

所谓指令相关，是指流水线中，如果某指令的某个阶段必须等到它前面的另一条指令的某个阶段完成才能开始，也即是两条指令之间存在着某种依赖关系，则两条指令存在指令相关。

指令相关包括结构相关、数据相关、和控制相关。流水线冲突/冒险（Hazzard）是指流水线中由于指令相关的存在，导致流水线中出现“断流”或“阻塞”，下一条指令不能在预期的时钟周期加载流入到流水线中。流水线冲突包括结构冲突、数据冲突和控制冲突。

1） 数据冲突

后续指令要用到前面指令的操作结果，而这个结果尚未产生或尚未送到指定的位置，从而造成后序指令无法继续执行的状况，称为数据冲突。根据指令读访问和写访问的顺序，常见的数据冲突包括先写后读冲突（Read after write，RAW）、先读后写冲突（Write after read，WAR）、写后写冲突（Write after write，WAW）。假定连续的两条指令 i 和 j，其中第 i 条指令在第 j 条指令之前流入到流水线，两条指令之间可能引起的数据冲突如下：

(1)先写后读冲突 RAW

如果第 j 条指令的源操作数是第 i 条指令的结果操作数，这种数据冲突被称之为先写后读冲突（RAW）。当指令按照流水的方式执行的时候，由于指令 j 要用到指令 i 的结果，如果指令 j 在指令 i 将结果写入寄存器之前就在译码阶段读取了该寄存器的旧值，则会导致读取数据出错。

(2)先读后写冲突 WAR

如果第 j 条指令的结果操作数是第 i 条指令的源操作数，这种数据冲突被称之为先读后写冲突（WAR）。当指令 j 去写该寄存器的时候，指令 i 已经读取过该寄存器，所以这种数据相关对指令的执行不构成任何影响。

(3)写后写冲突 WAW

如果第 j 条指令和第 i 条指令的结果操作数是相同的，这种数据冲突被称之为写后写冲突（WAW）。当指令按照流水的方式执行的时候，如果第 j 条指令的写操作发生在第 i 条指令的写操作之后，这种 WAW 冲突对指令的执行没有影响；但在乱序调度的流水线中，有可能第 j 条指令的写操作发生在第 i 条指令的写操作之前，此时会写入顺序错误，结果单元中留下的第 i 条指令的执行结果，而不是第 j 条指令的执行结果。由于本实验并未采用乱序发射技术，所以 WAW 冲突在本实验中不予考虑。

正常的程序都会存在着较多的 RAW 数据相关，为了有效的避免结果出错，最简单的处理方法，就是推后执行与其相关的指令，来保证指令或程序执行的正确性，如果利用软件方法解决就是在存在数据相关的指令之间插入空指令，这需要编译器支持。如果采用硬件方法解决就是所谓插入“气泡”的方法，译码 ID 阶段的指令如果与 EX、MEM 或 WB 阶段的指令存在数据相关，则译码阶段的指令暂停一个时钟周期，时钟到来时将 ID/EX 流水接口部件清空，也就是在执行阶段插入一条空指令（气泡），如插入一个气泡数据相关性还不能消失，则继续插入气泡直至数据相关消失，插入气泡的个数与具体设计相关。

2） 结构冲突

由于多条指令在同一时钟周期都需使用同一操作部件而引起的冲突被称为结构冲突。假如流水线只有一个存储器，数据和指令都放在同一个存储器中。当 load 指令进入访存 MEM 阶段时，而取指令阶段也同时需要访问存储器取出新的指令，这时就会发生访存冲突。这样的结构冲突实际上在单周期 CPU 中就存在，解决方法是采用独立的指令存储器和数据存储器（哈佛结构），现代 CPU 中指令 cache 和数据 cache 分离也是这种结构，流水线中也可以采用同样的

解决方案。另外一个方案是在流水线中插入一个“气泡”，使得取指令阶段暂时停顿一个时钟周期，下一个时钟周期到来时进入译码阶段的是一个气泡操作（空指令），等到 load 指令访存操作结束以后，取指令阶段再次重新启动，相比哈佛结构这种方案会使得流水线暂停一个时钟周期，引起性能的损失。

3） 控制冲突

当流水线遇到分支指令或其它会改变 PC 值的指令时，分支指令后续已经流入流水线的相邻指令可能不能进入执行阶段，这种冲突成为控制冲突，也称为分支冲突，由于分支指令跳转是否成功或改变后的 PC 值要等到执行 EX 阶段或访存 MEM 阶段才能确定或计算出来（具体哪个阶段与具体设计相关），所以分支指令后续的若干指令已经预取进入流水线（后续预取指令条数称为误取深度），当分支指令成功跳转时，流水线误取指令不能进入执行阶段，此时需要清空误取指令，同时修改 PC 的值，取出正确的跳转分支位置的指令。发生控制冲突时，流水线会清空误取指令，浪费了若干时钟周期，引起流水线性能降低。

还可以采用基于软件或硬件方法的分支预测，提前预取正确的指令，以尽量减少由于控制相关所导致的对流水线性能的影响。

1.3.3 实验内容

进一步改造理想流水线 MIPS CPU，增加数据相关检测逻辑，增加插入气泡逻辑，增加流水暂停逻辑，增加分支冲突处理逻辑，使得该流水线能处理数据冲突，控制冲突，资源冲突等所有流水冲突，并能正确运行单周期测试程序 benchmark.hex。

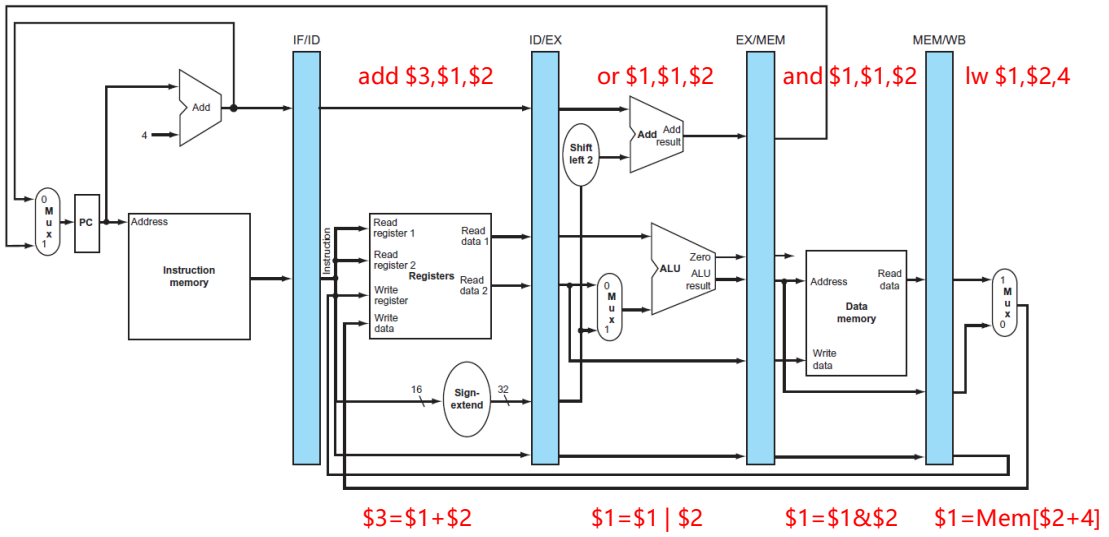

图 1.10 五段流水 MIPS CPU 中的数据相关

（1）**数据相关检测**，由于译码 ID 段中需要取操作数，所以数据相关检测逻辑应该设置在 ID 段，如图 1.10 所示，由于数据只能在写回 WB 阶段写入，所以 ID 段正在处理的 ADD 指令与执行 EX 段，访存 MEM 段，写回 WB 段 3 段的指令都存在数据相关。本实验需要实验者设计数据

相关检测逻辑检测这种相关性，当存在数据相关时，可进行插入气泡的处理。数据相关的依据是比较当前指令的源操作数是否和后续段的目的操作数是否相同。注意在 MIPS 指令集中不同指令所包含的源操作数是不同的，如 R 型指令涉及两个源操作数 Rs,Rt，I 型指令涉及一个或两个源操作数 Rs,Rt，分支指令（Beq，Bne）涉及两个源操作数 Rs,Rt，J 型指令无源操作数，但会产生控制冲突。注意比较时可能需要和后面三段的目的操作数均做比较，且需要考虑后续三条指令是否存在目的操作数。

（2）**数据相关处理**，图 1.10 中译码 ID 段与写回 WB 段的数据相关可以通过先写后读的方式解决，具体方式是数据写入寄存器文件时采用下跳沿写入，由于寄存器的读出是组合逻辑，数据写入即刻获得寄存器新值，所以只要在下跳沿成功写入后，上跳沿触发流水线进行指令传送时，送入 ID/EX 段的操作数已经是最新的寄存器值。而 ID 段与 EX 段和 MEM 段的数据相关则只能通过插入气泡的方式解决。

（3）**插入气泡逻辑**，出现数据相关时将在 EX 段插入气泡，IF 段，ID 段暂停。图 1.11 中译码 ID 段与执行 EX 段存在数据相关，假设相关检测逻辑已经检测到存在数据相关，应产生 stall（暂停）控制信号，暂停取指 IF 段以及译码 ID 段的工作以延缓取操作数的执行（可以通过控制程序计数器和 IF/ID 流水接口部件的使能端实现,这样时钟到来时锁存器的值不会改变，可能需要增加新的引脚），另外下一个时钟上跳沿到来时，需要向执行 EX 段插入一个气泡（空指令，MIPS 中空指令是全零的机器码，功能是 0 号寄存器左移零位送入 0 号寄存器），以避免译码 ID 段的指令向后传送。插入气泡可以通过 ID/EX 流水接口部件清零完成，为此需要为流水接口部件增加清零引脚。

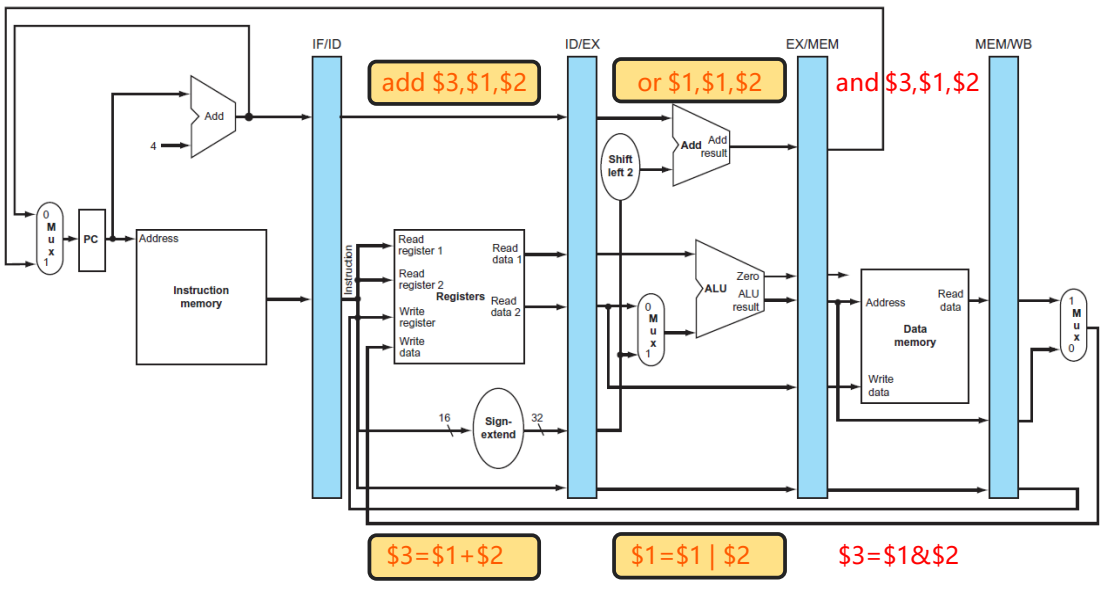

图 1.11 五段流水 MIPS CPU 中的数据相关---插入气泡前

时钟上跳沿到来后，将在 EX 段插入一个气泡，如图 1.12 所示，但此时数据相关检测逻辑还可以检测到 ID 段和 MEM 段存在数据相关，此时继续重复前面的逻辑，在 EX 段插入气泡，暂停 IF 和 ID 段。

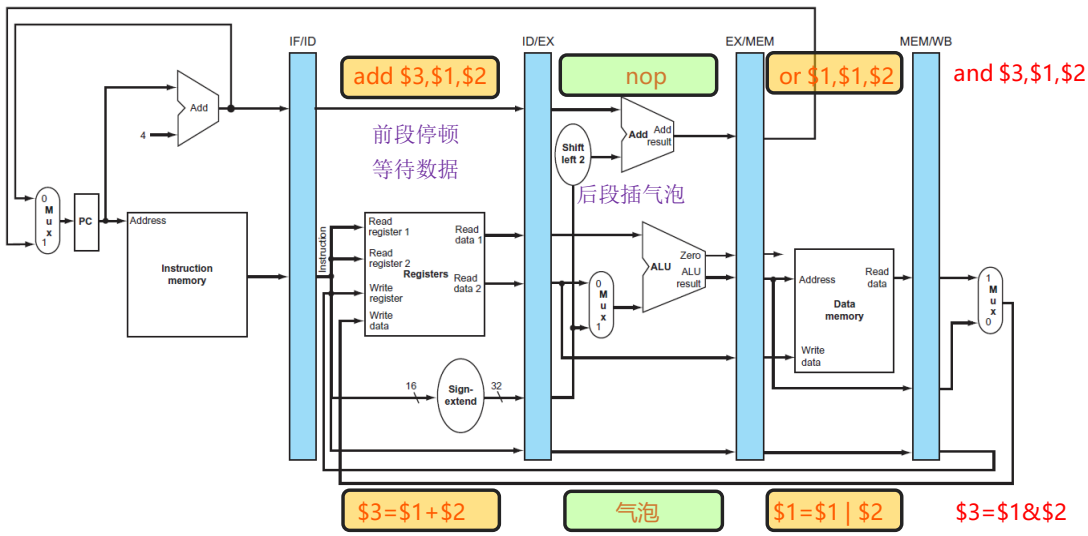

图 1.12 五段流水 MIPS CPU 中的数据相关---插入气泡后

下一个时钟上跳沿到来后，将在 EX 段继续插入一个气泡，如图 1.13 所示，此时数据相关性消失，流水线暂停信号 Stall 自动撤除，流水线重新恢复正常。

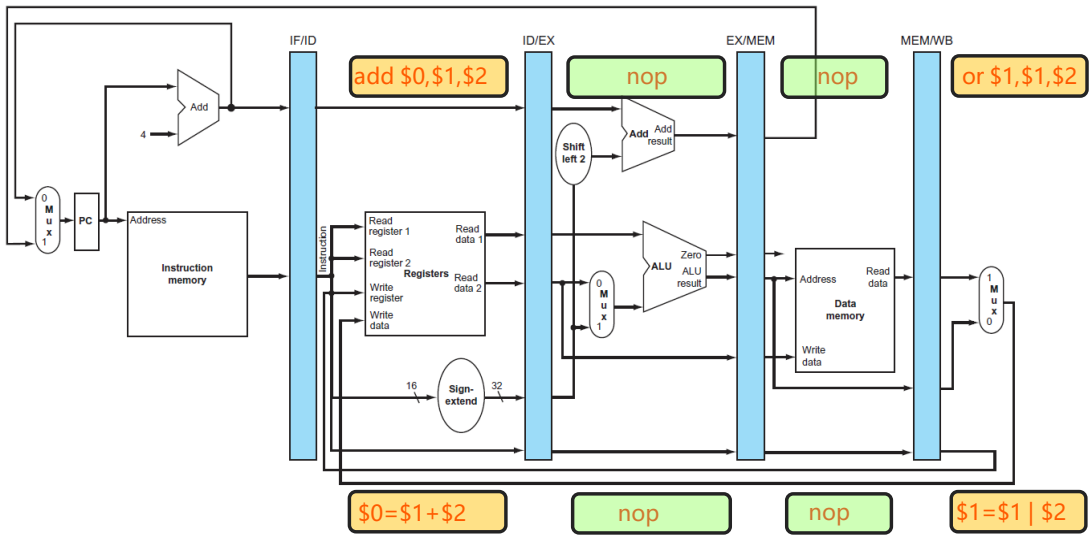

图 1.13 五段流水 MIPS CPU 中的数据相关---数据相关消除

下一个时钟上跳沿到来时，译码 ID 段处理新的指令，存在数据相关的加法指令通过 ID/EX 流水接口部件进入执行 EX 段，如图 1.14 所示。

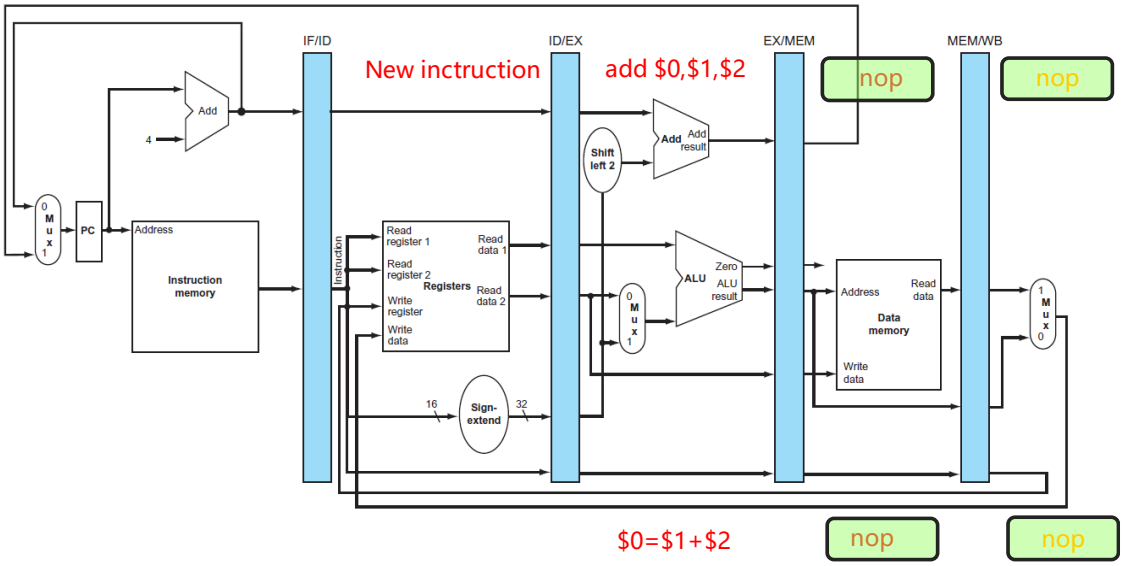

图 1.14 数据相关消除后的执行状态

（4）**结构冲突处理**，当多条指令在同一时钟周期都需使用同一操作部件而引起的冲突被称为结构冲突，在流水线设计中也会存在各种结构冲突，如计算 PC+4，计算分支地址，运算器运算都需要使用运算器，访问指令和访问内存都需要使用存储器，解决方案是增设加法部件避免运算冲突，增设指令存储器避免访存冲突。另外译码阶段读寄存器与写回阶段写寄存器也存在结构冲突，这里可以采用先写后读的方式解决（下跳沿写入，与流水接口时钟触发相反）。

（5）**控制冲突处理**，如图 1.15 所示，在执行阶段出现了一条无条件分支指令 J 指令，这里假设无条件分支指令在 EX 段执行，J 指令执行时需要修改 PC 的值，运算器运算的分支地址的结果将传输经多路选择器传送给 PC，下一个时钟上跳沿到来时锁存进入 PC，ID 段的 ADD 以及 IF 段的 SUB 指令都属于误取进入流水线的指令，都无法继续在流水线中执行，需要清除。

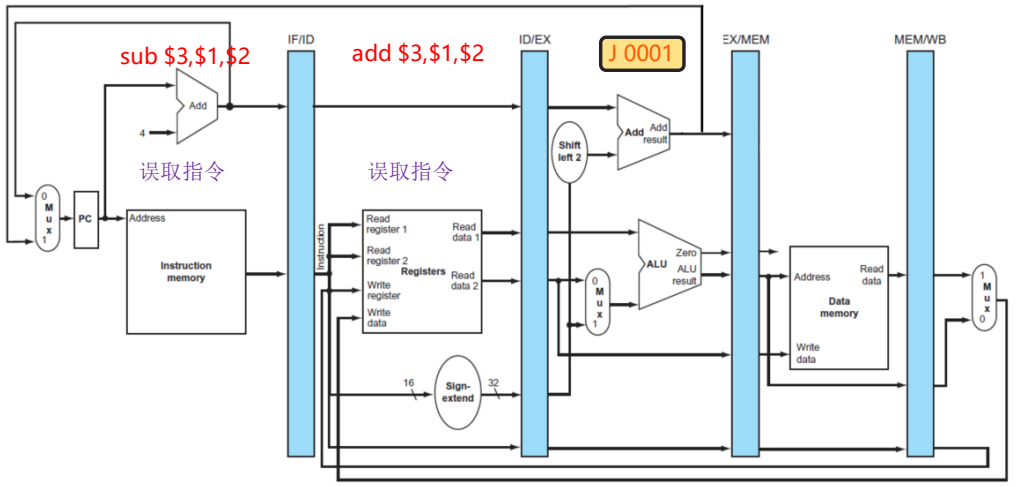

图 1.15 五段流水 MIPS CPU 中的分支相关

图 1.16 时钟上跳沿到来之后流水线的状态，从图可看出，IF/ID 接口部件以及 ID/EX 接口部件所有数据和控制信号全部被清零，全零的 MIPS 指令是 sll $0,$0,0 指令，不会改变 CPU 状态，等同与空操作 nop，成功插入气泡，所以这里控制冲突时应该给出 IF/ID、ID/EX 段的清空信号，时钟上跳沿到来是完成误取指令清空。

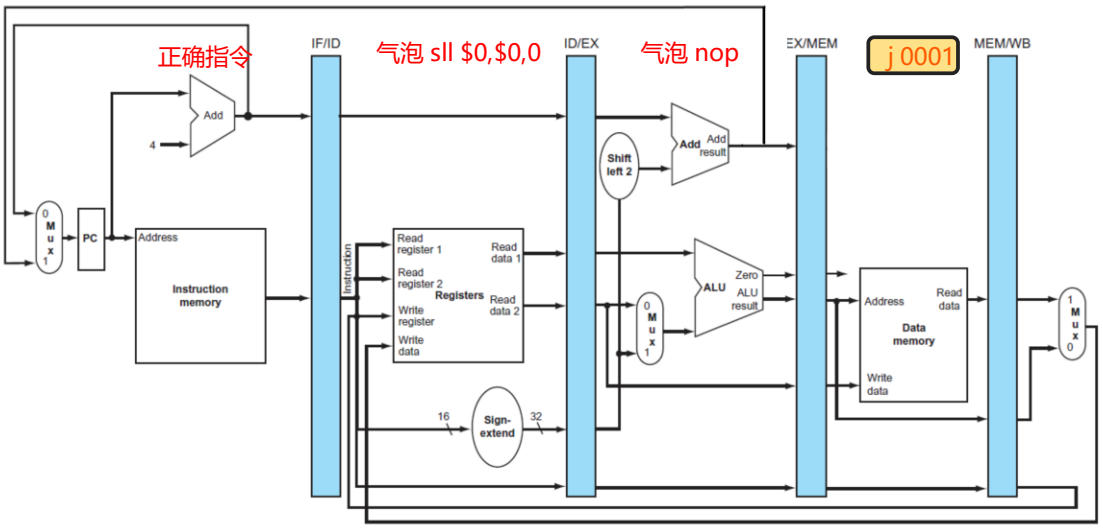

图 1.16 五段流水 MIPS CPU 中的分支相关正确处理

注意分支指令的执行也可安排在 ID 段，MEM 段，甚至 WB 段完成，不同段执行误取深度不一样，在执行 EX 段进行分支处理的误取深度为 2，误取深度越大，对流水性能造成的影响也越大，实验时可以酌情考虑在哪个阶段完成分支处理。

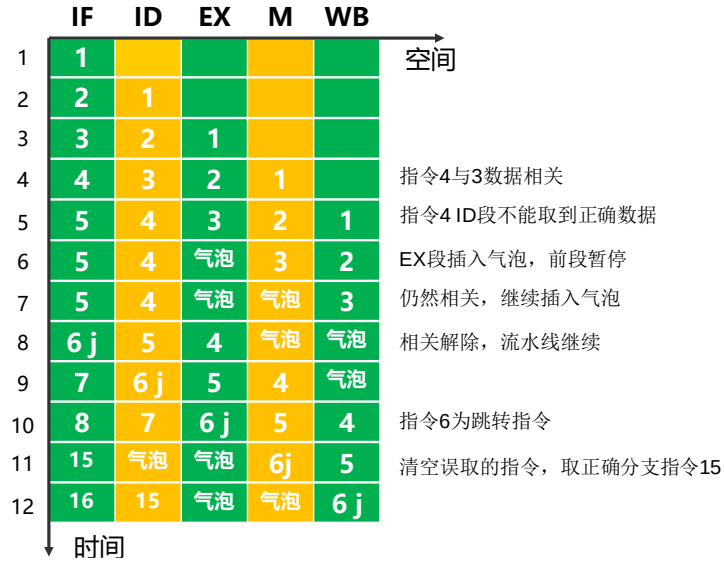

图 1.17 存在数据相关和分支相关的流水线时空转置图

图 1.17 为 MIPS 五段流水 CPU 处理数据相关和分支相关的流水线时空转置图，此图和普

通的时空图有区别，横坐标是空间，代表各功能部件段，纵坐标是时间，每一格代表一个时钟周期，图中单元格的数字代表流入流水线功能部件的指令序号，采用这种方法展示时空图更有利于大家结合具体电路设计进行分析验证。

1.3.4 实验步骤

(1) **设计相关检测逻辑**，分析实验中需要支持的所有 mips 指令的指令格式，根据需要设计数据相关检测逻辑，用于判断 ID 段的指令与 EX 以及 MEM 段指令之间的相关性。相关检测逻辑封装完成后设计简单测试程序进行测试。
(2) **改造流水接口部件，**使其能实现流水暂停以及插入气泡的功能。
(3) **实现插入气泡逻辑，**在 ID 段的控制器中实现插入气泡逻辑，并使用简单测试程序进行测试。
(4) **实现控制相关逻辑**，增加相关逻辑，是的无条件分支指令，有条件分支指令能在流水线上正确运行，并使用简单测试程序进行测试。
(5) **流水综合调试，**加载标准测试程序进行功能调试，注意统计时钟周期数。
(6) **提交教师检查。**

1.3.5 实验思考

(1) 插入气泡能否直接使用寄存器的异步清零信号，为什么？
(2) MIPS 零号寄存器是比较特殊的寄存器，如果源操作数是零号寄存器时是否存在数据相关？你是如何解决的？气泡指令之间是否存在相关？
(3) 采用气泡方式解决数据相关后测试标准测试程序 benchmark 时钟周期为什么反而比单周期 CPU 多了很多？

1.4 重定向流水线

1.4.1 实验目的

学生理解数据重定向的基本原理，理解 Load-use 相关的处理机制，能够在气泡流水线 CPU 的基础上增加相应的逻辑功能部件，使得除了 Load-use 相关的所有数据相关都可以采用重定向方式进行处理，最终实现的 CPU 能正确运行单周期 CPU 实验中测试过的 benchmark 标准测试程序。

1.4.2 背景知识

气泡流水线通过延缓译码 ID 段取操作数动作的方式解决数据相关问题，但大量气泡的插入会严重影响流水性的性能，还有一种思路是先不考虑译码阶段所取的操作数是否正确的问题，而是等到执行阶段实际需要使用这些操作数时再考虑正确性问题，如图 1.18 所示，执行

EX 段的 or 指令与访存段，写回段的两条指令均存在数据相关，此时执行段取得的数据应该是错误的数据，正确的数据分别存放在 EX/MEM 以及 MEM/WB 流水接口部件中，还未来得及写回到寄存器中，此时可以直接将正确数据从对应位置重定向（Forwarding）到运算器 ALU 的操作数端（也称为旁路 Bypass），这样就可以避免插入气泡引起的流水线性能下降，重定向方式可以解决大部分的数据相关问题，可大大优化流水线性能。

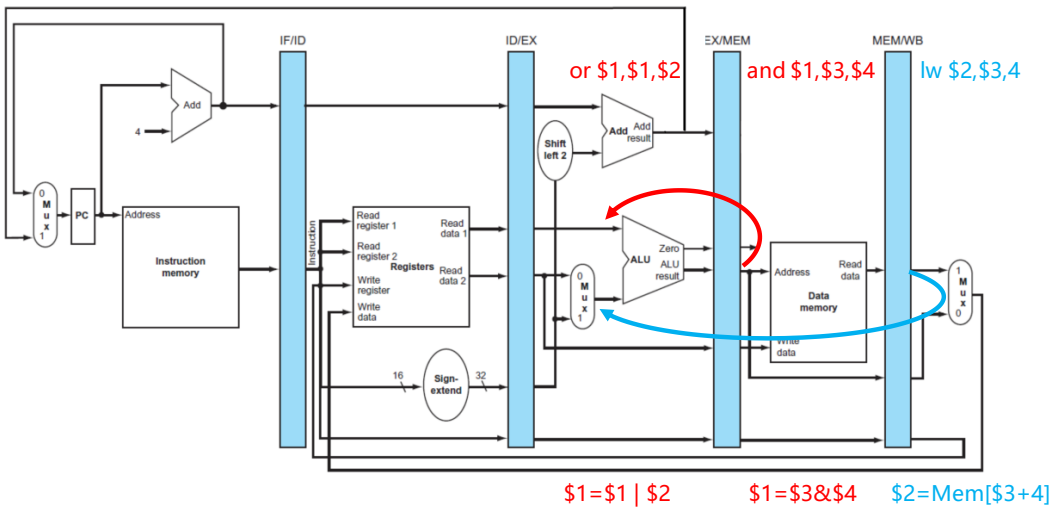

图 1.18 数据重定向示意图

但是如果出现相邻两条指令存在数据相关，且前一条指令是访存指令时（称为 Load-Use 相关），不能采用重定向方式进行处理，如图 1.19 所示，执行段 or 指令需要使用 1 号寄存器，而访存阶段目的寄存器也是 1 号寄存器，这时 1 号寄存器的值并没有锁存在 EX/MEM 中，而是必须等待数据存储器读操作完成后才会出现在 Read data 引脚上，如果将该引脚直接重定向到 ALU 端，逻辑上也可以成立，甚至在 Logisim 以及 FPGA 上也可以成功实现，但实际上这样会使得 EX 段的延迟变成了访存延迟+运算器延迟，违背了流水线分段尽量使各段延迟相等的原则，会使得系统最大时钟频率降低一倍。

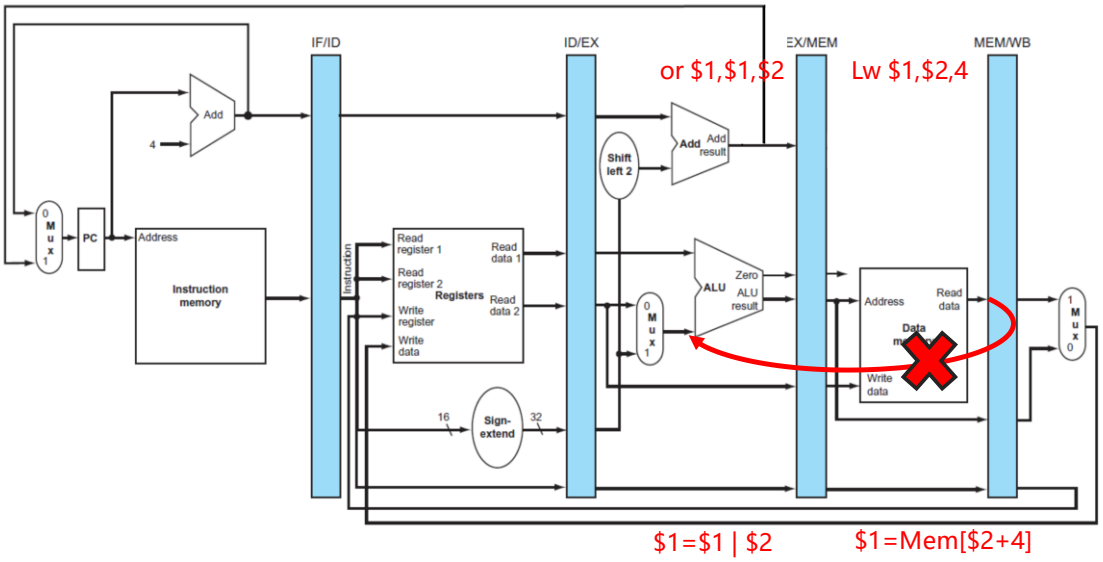

图 1.19 Load-Use 相关

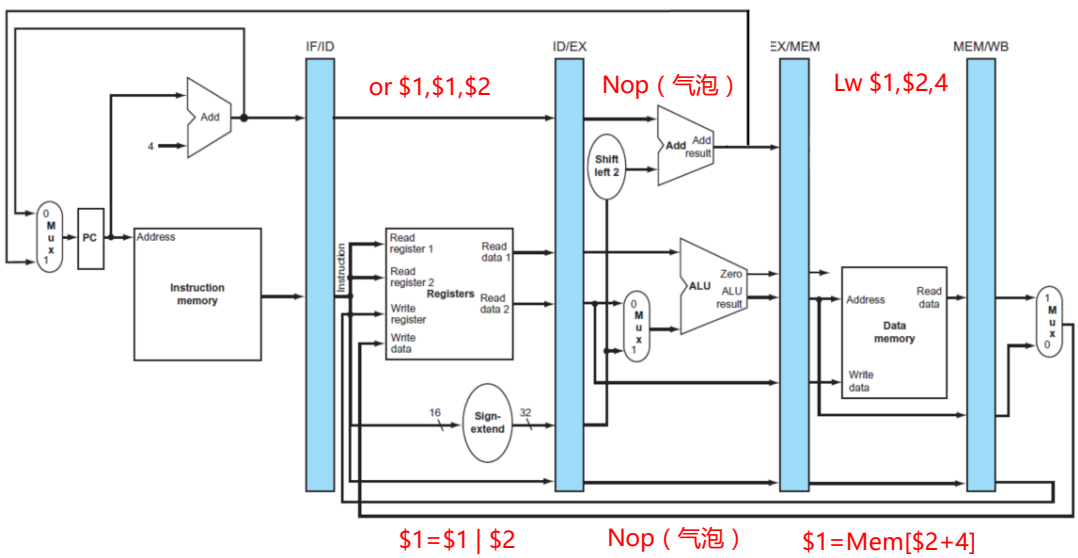

图 1.20 插入气泡消除 Load-Use 相关

所以对于 Load-Use 数据相关，应在译码阶段就及时检出，并强制插入一个气泡，消除这种相关，如图 1.20 所示。

1.4.3 实验内容

进一步改造气泡式流水线 MIPS CPU，增加重定向机制，增加 Load-Use 数据相关检测机制，使得流水线能在不插入气泡的情况下处理大部分数据相关问题，最终能正确运行单周期测试

程序 benchmark.hex。

1.4.4 实验步骤

（1）**改造运算器两输入端的多路选择器**，增加重定向的若干数据通路，如 EX/MEM 中锁存的运算器运算结果、MEM/WB 中锁存的运算器运算结果、MEM/WB 中锁存的访存数据等，重新增加运算器操作数选择控制信号。

（2）**构造重定向逻辑，首先**改造数据相关检测逻辑，增加 Load-Use 数据相关检测逻辑，如出现 Load-use 相关执行原有插入气泡的逻辑进行处理。如出现非 Load-Usede 数据相关，则由重定向逻辑直接生成操作数来源选择信号，以便控制运算器操作数端的多路选择器。这里重定向逻辑可以放置在 ID 段，也可以放置在 EX 段，二者各有利弊，大家可自行权衡。

（3）流水综合调试，加载标准测试程序进行功能调试，注意统计时钟周期数。

（4）提交教师检查。

1.4.5 实验思考

(1) 重定向逻辑放在流水线哪个阶段更好，为什么？