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。

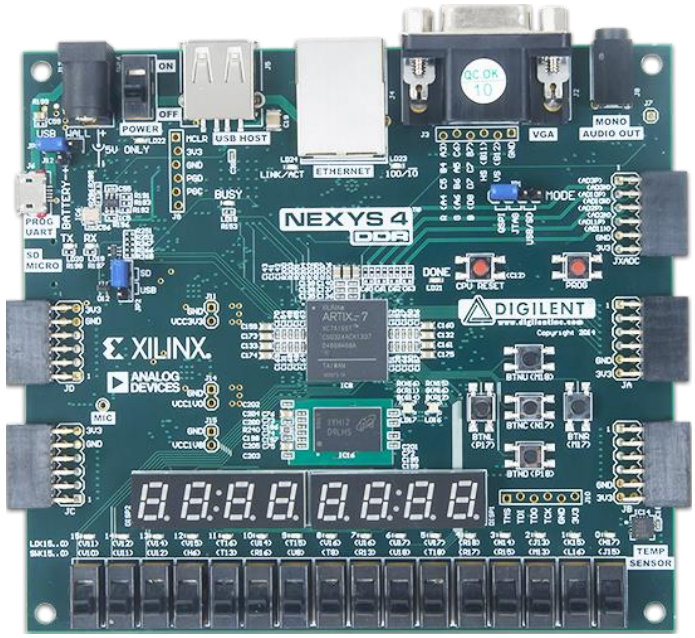

图 1.1 NEXYS 4 开发板

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。

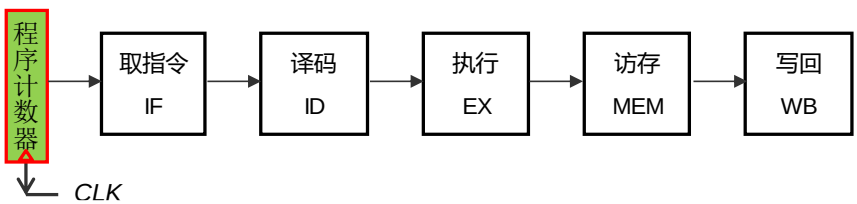

图 1.2 单周期 CPU 逻辑架构

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。

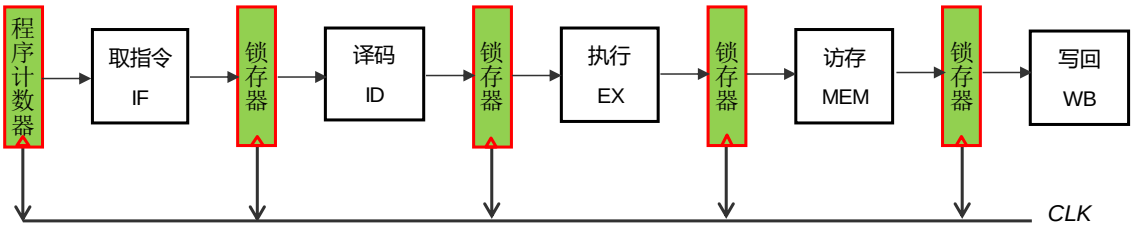

图 1.3 指令流水线逻辑架构

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。

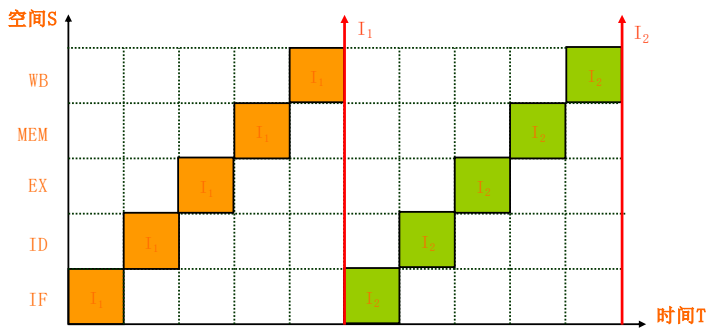

图 1.4 单周期 MIPS CPU 时空图

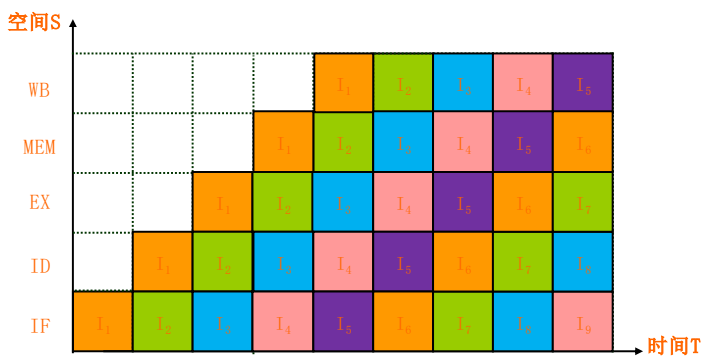

图 1.5 五段流水 MIPS CPU 时空图（无冲突）

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。

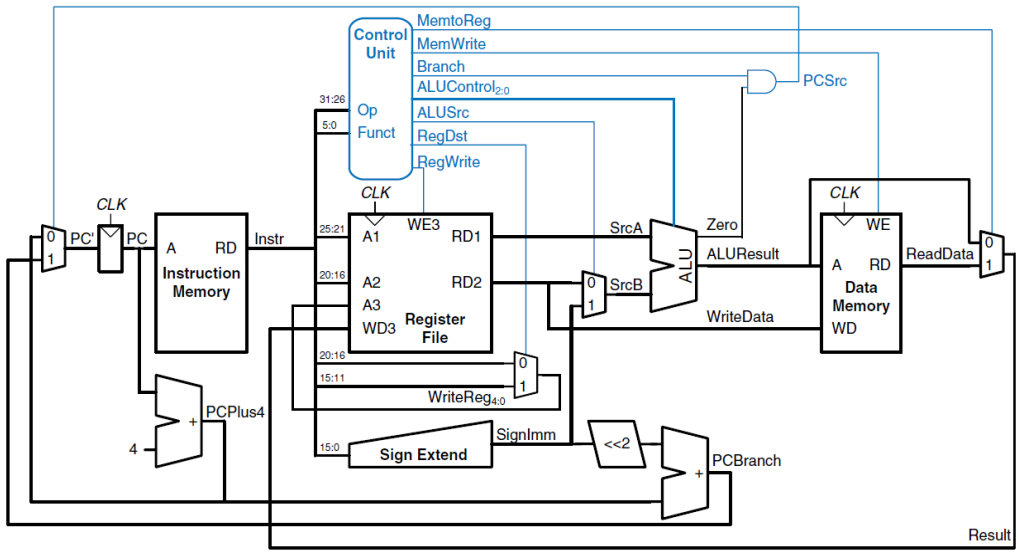

图 1.6 单周期 MIPS CPU

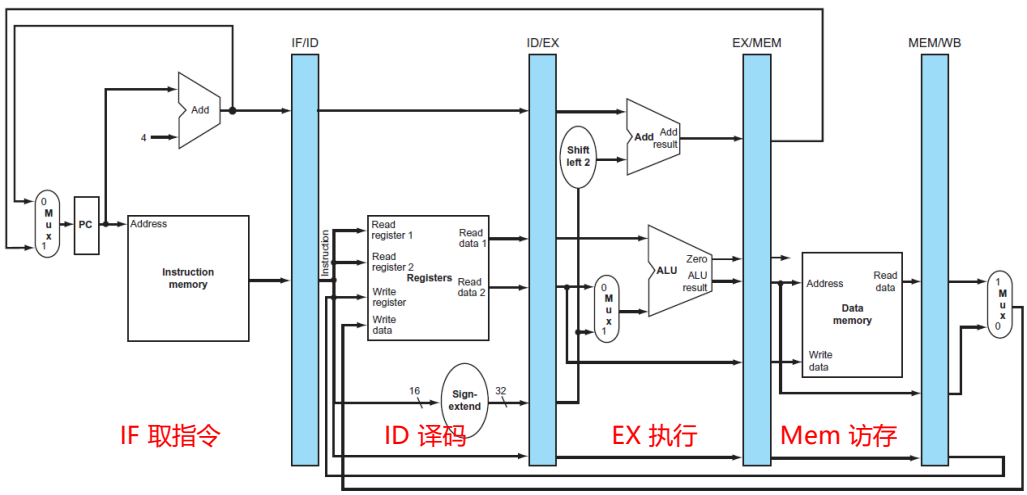

图 1.7 五段流水 MIPS CPU

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。

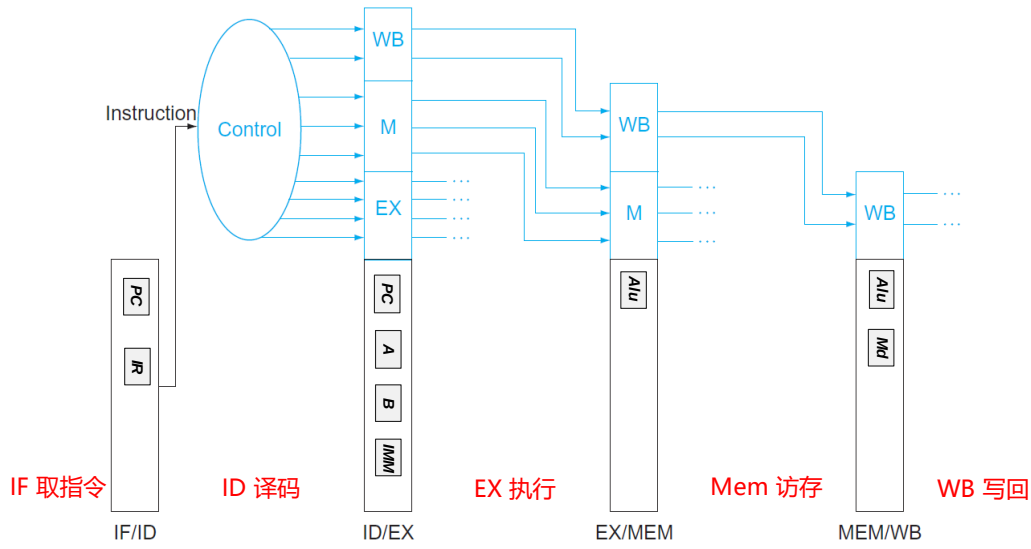

图 1.8 五段流水 MIPS CPU 数据与控制信号传递

五段流水 CPU 输入输出引脚如 XX 所示，请按相关要求进行流水线 CPU 设计，具体设计时请遵循如下注意事项。

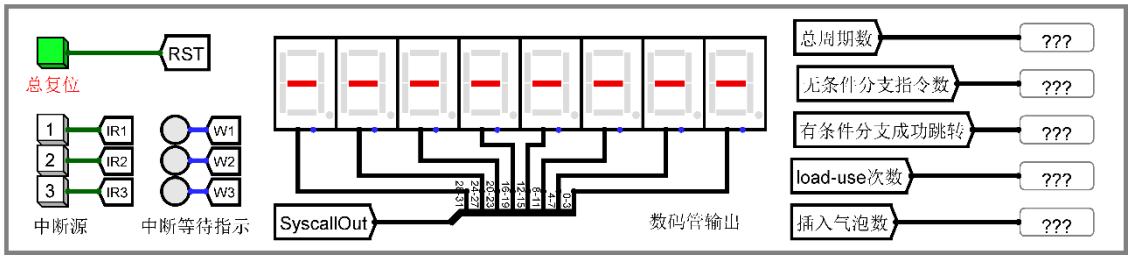

图 1.9 流水 CPU 输入输出引脚示意图

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 1.10 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。

(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) 控制信号可以按使用阶段用分线器汇总成 EX 段、MEM 段、WB 段三根控制总线，这样绘图时连线较为简单，布局更清晰。

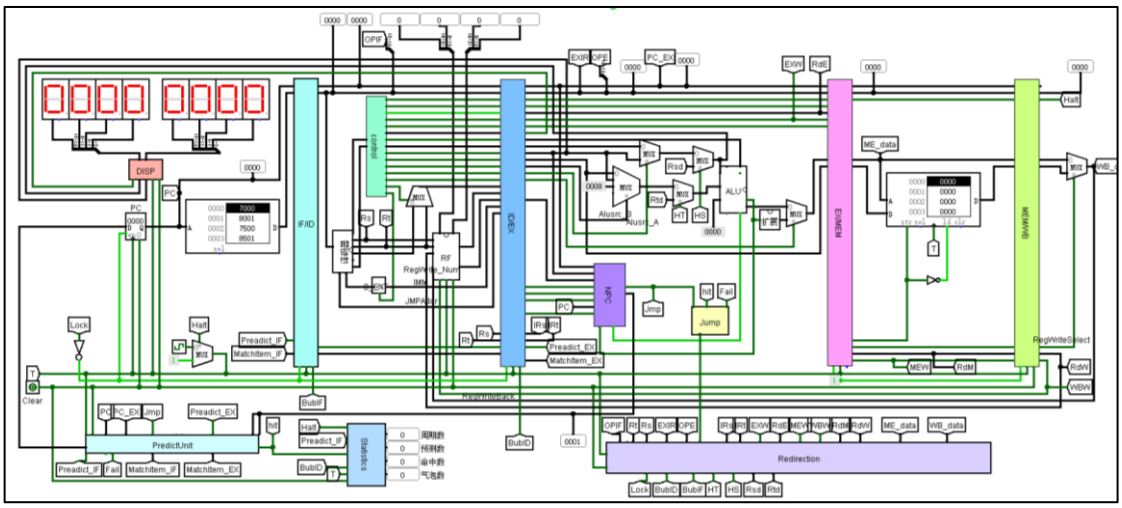

图 1.10 五段流水 MIPS CPU 原理图布局

(4) 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。

(5) 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。

(6) 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。

(7) 可以使用任何 logisim 内建的任何电路组件构建原理图。

(8) **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，注意最终理想流水线测试程序会在内存中写入数据，请自行检查后再提交检查。

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。

（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 1.11 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。

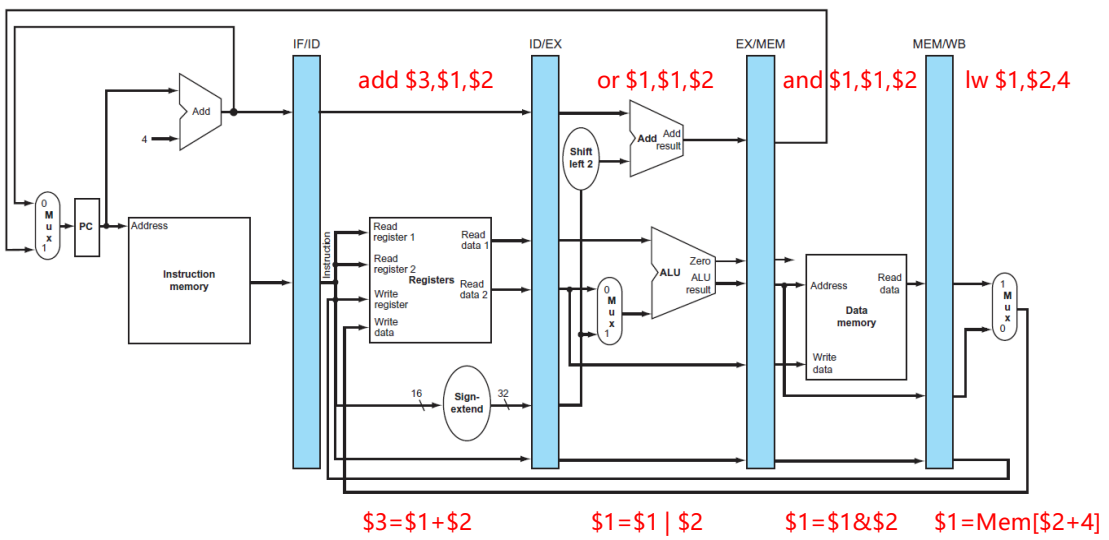

图 1.11 五段流水 MIPS CPU 中的数据相关

（2）**数据相关处理**，图 1.11 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 1.12 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。

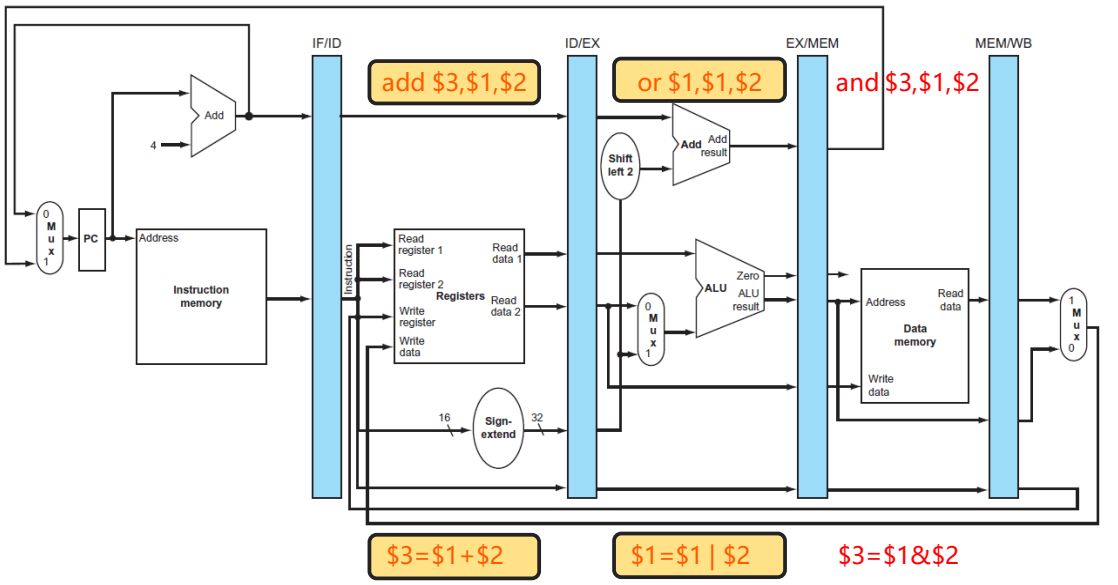

图 1.12 五段流水 MIPS CPU 中的数据相关---插入气泡前

时钟上跳沿到来后，将在 EX 段插入一个气泡，如图 1.13 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。

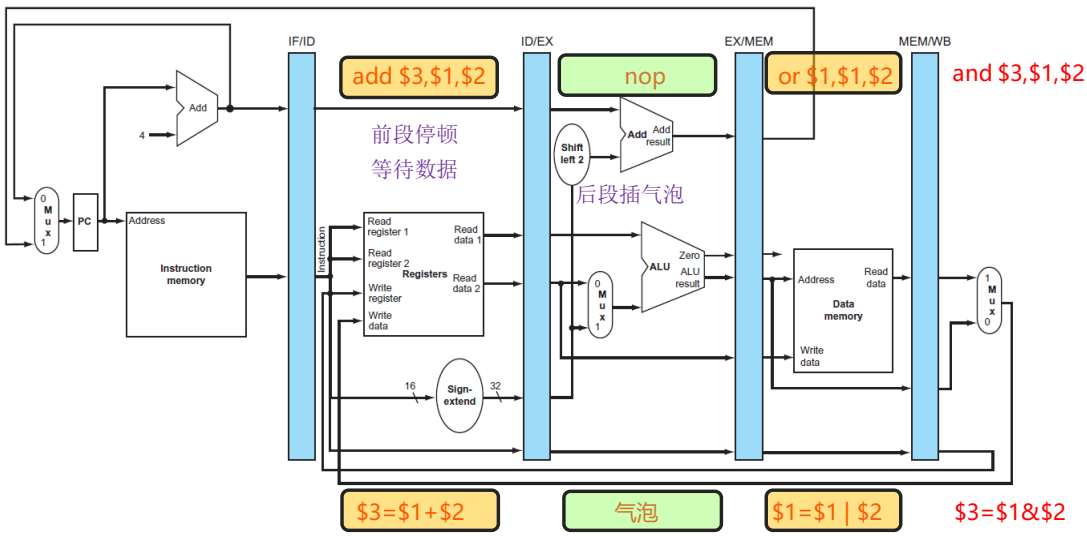

图 1.13 五段流水 MIPS CPU 中的数据相关---插入气泡后

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 1.14 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。

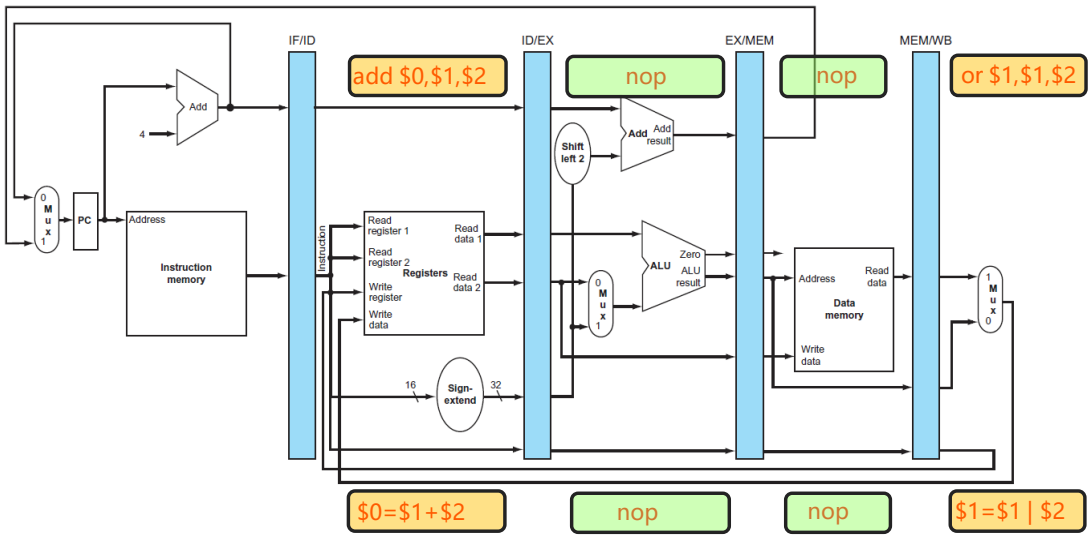

图 1.14 五段流水 MIPS CPU 中的数据相关---数据相关消除

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 1.15 所示。

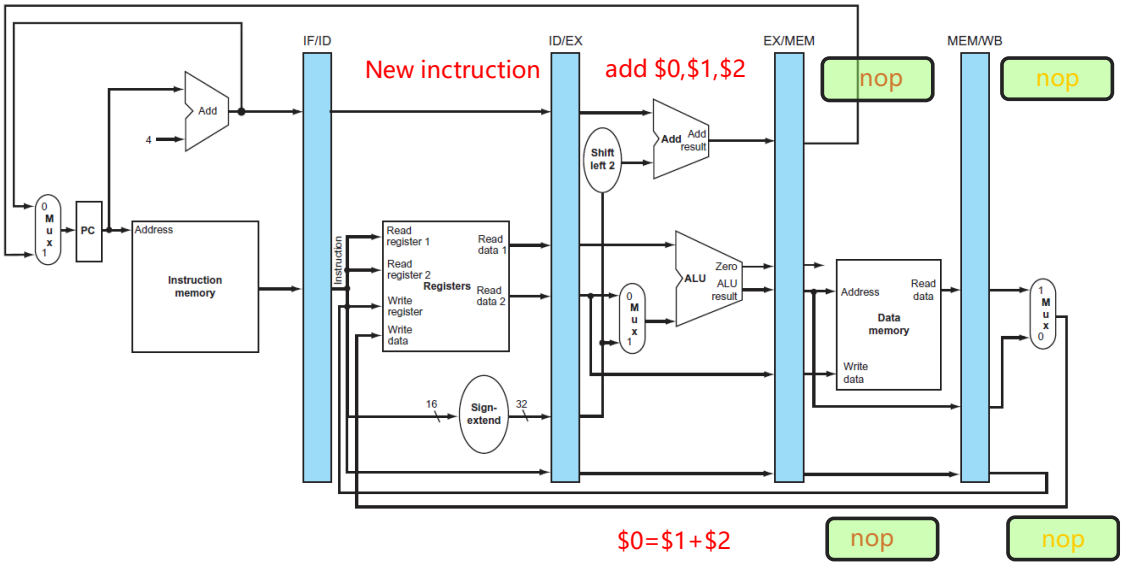

图 1.15 数据相关消除后的执行状态

（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 1.16 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。

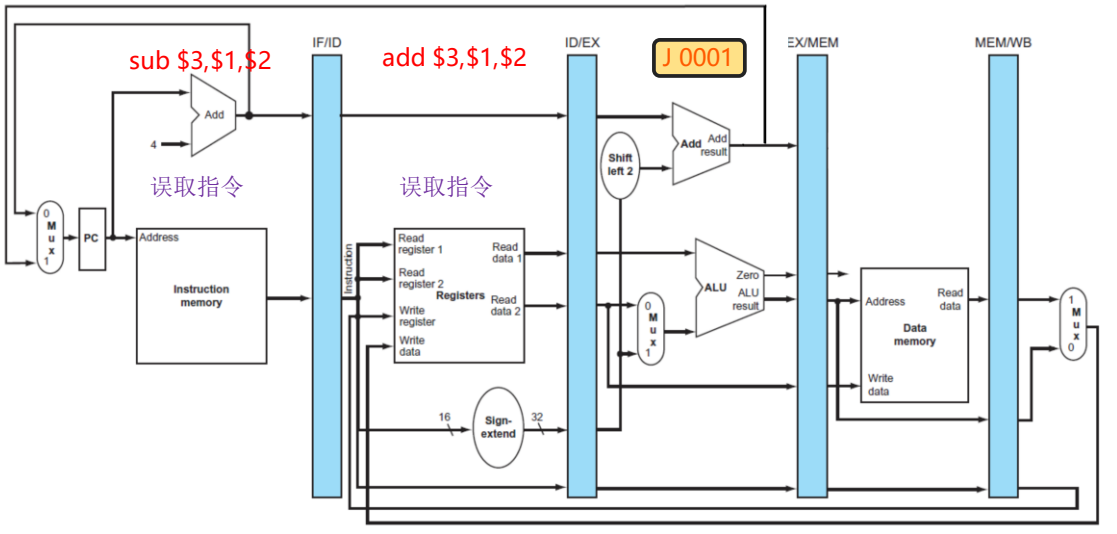

图 1.16 五段流水 MIPS CPU 中的分支相关

图 1.17 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。

注意：真实 MIPS CPU 中采用了分支延迟槽技术，也就是分支指令后面的一条指令也会被执行，为简化实验，我们取消分支延迟槽技术，所以 MIPS 指令规范中所有 PC+8 的要求实验中应该全部改成 PC+4。

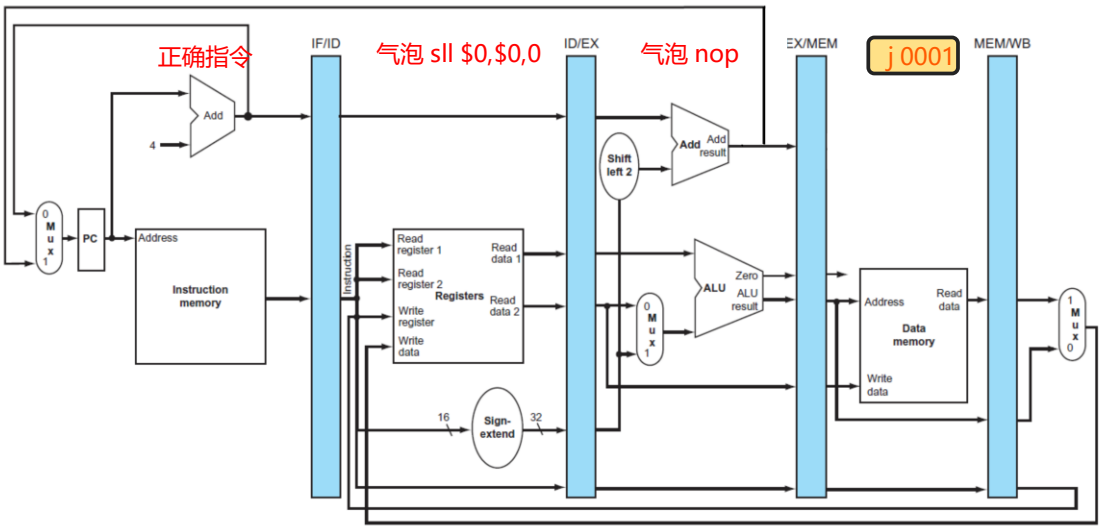

图 1.17 五段流水 MIPS CPU 中的分支相关正确处理

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 1.18 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。

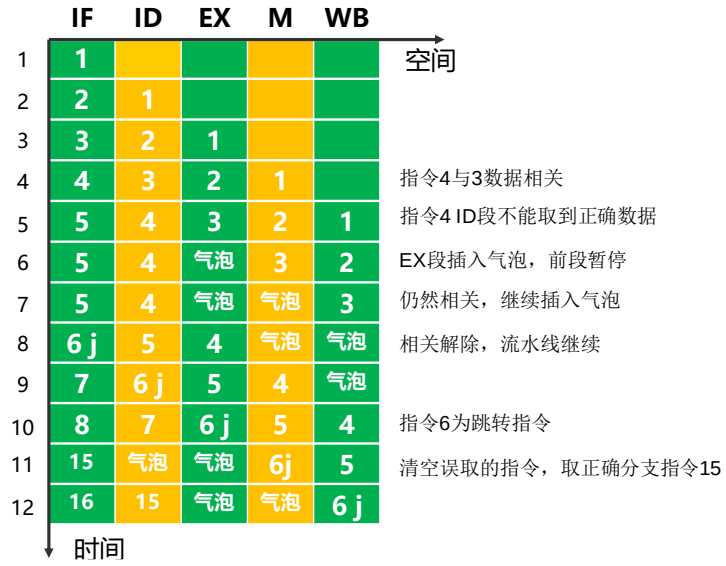

图 1.18 存在数据相关和分支相关的流水线时空转置图

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) 插入气泡的个数需要用电路进行控制吗？为什么？

(3) 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？

(4) 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 1.19 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。

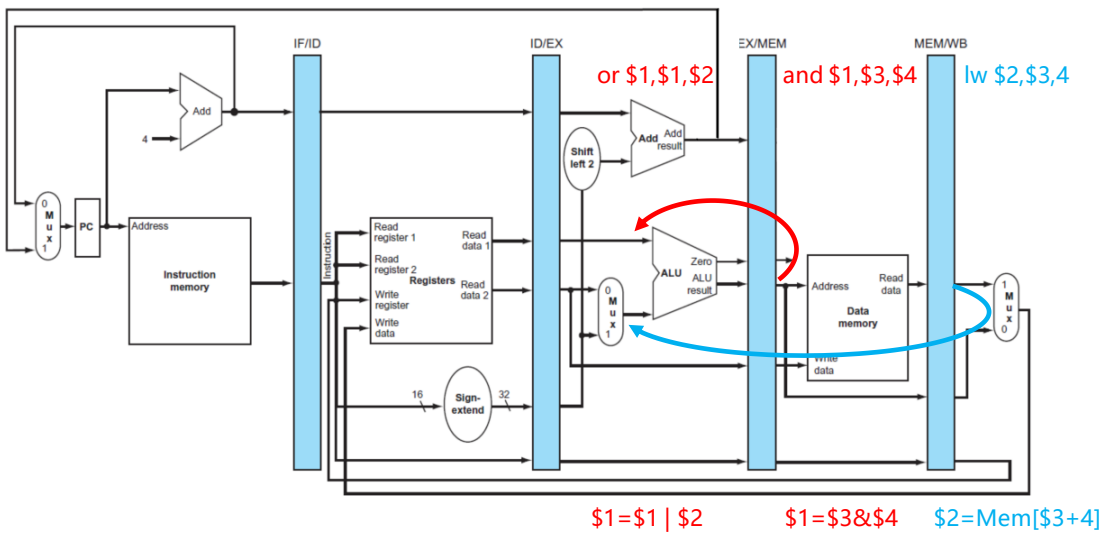

图 1.19 数据重定向示意图

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 1.20 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。

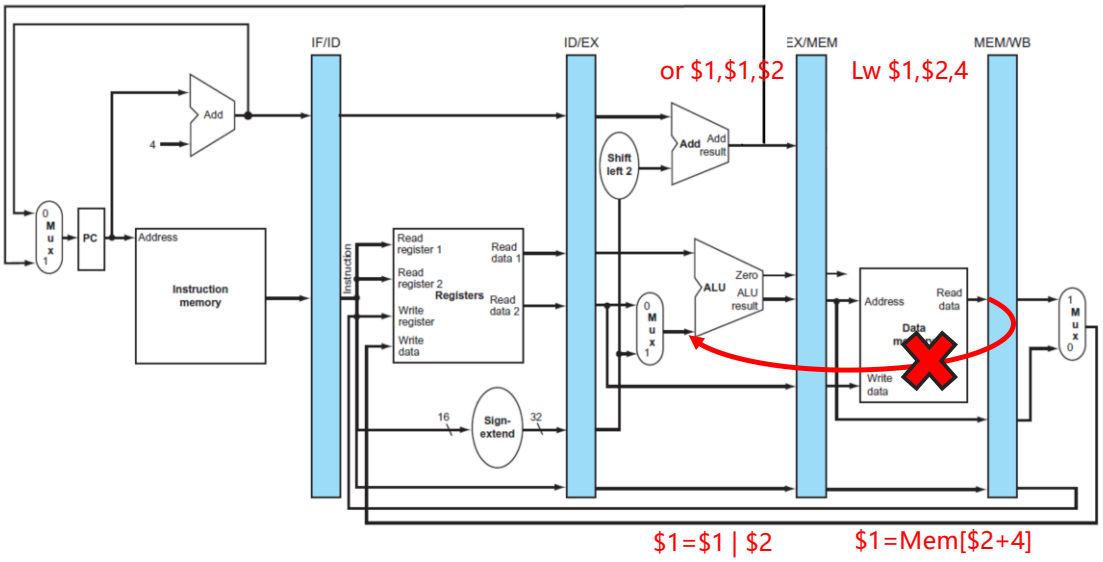

图 1.20 Load-Use 相关

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 1.21 所示。

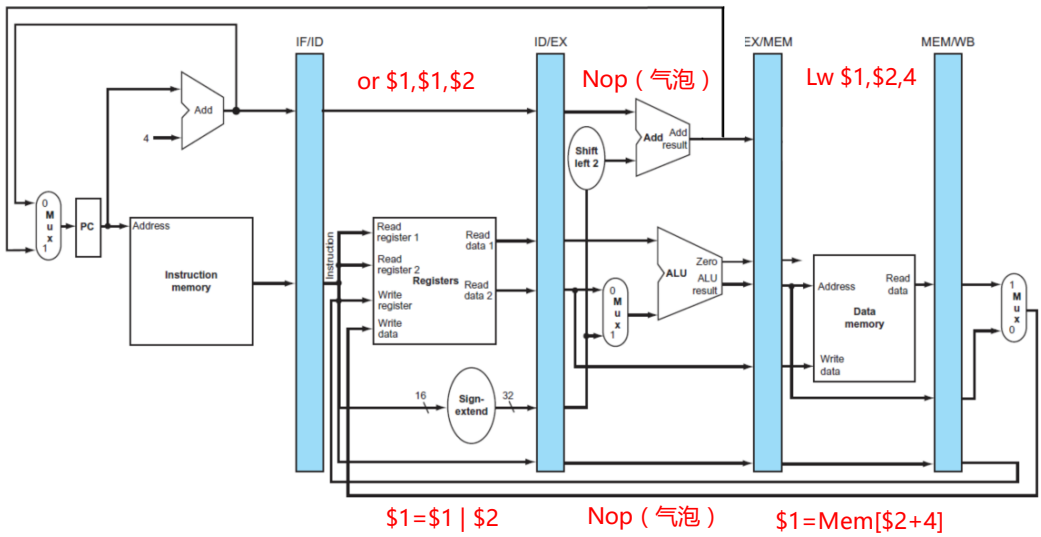

图 1.21 插入气泡消除 Load-Use 相关

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？

1.5 流水 CPU 上板实验

1.5.1 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流水 CPU 在该平台上具体实现，并能正确运行标准测试程序。

1.5.2 实验内容

将支持重定向机制的五段流水 MIPS CPU 从 Logisim 移植到 N4-DDR FPGA 平台。

1.6 单级中断实验

1.6.1 实验目的

学生掌握单级中断处理机制，能在单周期 CPU 中设计单级中断处理机制，能处理多个外部中断事件，能正确的终止主程序的执行，转为为按钮事件服务的中断服务程序，中断服务程序执行完毕后应返回主程序继续运行，不同的按钮会进入不同的中断服务程序。

1.6.2 背景知识

计算机系统运行时，若系统外部、内部或当前程序本身出现某种非预期的事件，CPU 将暂时停下当前程序，转向为该事件服务，待事件处理完毕，再恢复执行原来被终止的程序，这个过程称为中断。产生中断的事件对 CPU 来说是随机发生的，中断技术把有序的程序运行和无序的中断事件统一起来，大大增强了计算机系统的处理能力和灵活性。中断是现代计算机普遍采用的一项重要技术，可以实现主机和外设并行工作，以提高了整个系统的工作效率。可以方便计算机进行程序调试、人机交互、故障处理、实时处理。

(1) 中断分类

根据引起中断的事件来自于CPU内部还是CPU外部，可分为内中断(也称为软中断)和外中断(也称为硬件中断)；根据进入中断的方式可分为自愿中断和强迫中断；根据其重要性可分为可屏蔽中断和不可屏蔽中断，如图1.22所示。

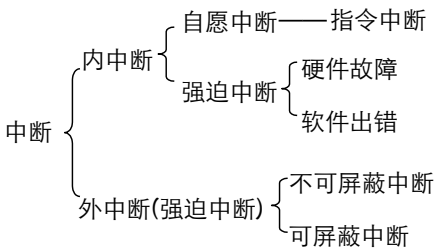

图 1.22 中断分类

(2) 中断优先级

当多个外部设备同时有中断请求时，就存在着优先级的问题，优先级高的先响应，优先级低的后响应，中断优先级分为相应优先级和处理优先级。响应优先级是指 CPU 对各设备中断请求进行响应的先后次序,它根据中断事件的重要性和迫切性而定,是在硬件电路上进行固定的，无法更改。处理优先级是指 CPU 实际完成中断处理程序的先后次序，可以通过中断屏蔽技术动态调整设备的处理优先级，如果不采用中断屏蔽技术，处理优先级和响应优先级相同。

(3) 中断嵌套

根据设备中断服务子程序能否被其他中断请求再次中断，可以分为单级中断和多重中断。

如果一个中断服务程序被执行，则 CPU 不响应其它中断请求（包括比本中断服务程序级别更高级的中断），而只有在该中断服务程序执行完成后才能响应其它中断请求，这种中断就是单级中断。在中断服务程序执行过程中，如果允许 CPU 响应其它中断请求，则这种中断称为多重中断，也称中断嵌套。中断嵌套中高优先级中断可以中断低优先级中断。

(4) 中断仲裁

同一时刻可能有多个设备同时发出中断请求，如何选择适当的设备进行中断响应，这就是中断仲裁需要解决的问题，中断仲裁和集中式与总线仲裁方式类似，仲裁方式与中断请求与 CPU 的连接关系有关，主要分为链式查询、独立请求、分组链式三种。

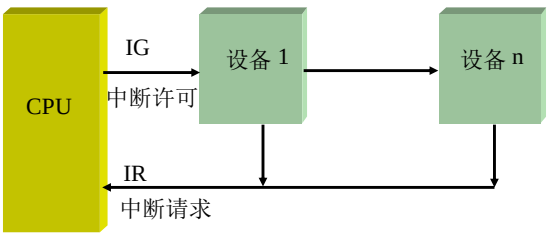

图 1.23 中断链式请求

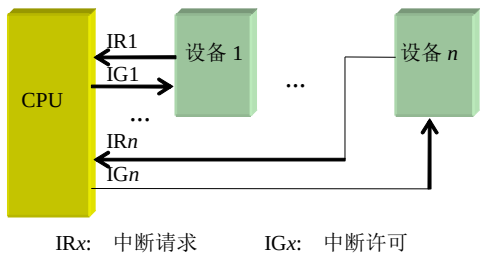

图 1.24 中断独立请求

链式请求方式中 CPU 只有一个中断请求输入 IR，所有设备都连接到 IR 引脚，需要发出中断时将 IR 置 1，CPU 收到中断请求 IR 后，通过中断许可 IG 信号对设备进行授权，授权信号 IG 先传递给设备 1，如设备 1 没有中断请求，将授权信号继续转发给设备 n，直至正确的中断设备。这种方式中断响应较慢，需要多个时钟周期才能实现中断授权，而且存在单点故障。

独立请求方式中每一个设备都有一组中断请求 IR 与中断授权 IG 信号与 CPU 相连，CPU 可以根据响应优先级直接进行中断响应，处理灵活，但 CPU 引脚过多，受限于引脚资源。实际系统中往往采用分组链式的折中结构，如图 1.25 所示。二维分组链式结构中每一组 IR，IG 信号被多个设备采用链式结构共享，称为中断共享。

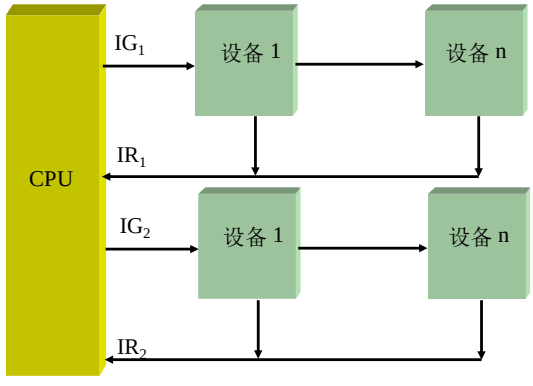

图 1.25 二维分组链式请求

(5) 中断识别

中断识别的任务是确定中断是由哪个中断源发出的，不同的中断请求连接方式对具体中

断源的识别是不同的，主要有程序查询、硬件查询和独立请求三种中断源识别的方法。独立请求方式识别最为简单，链式或分组链式相对较为复杂。以独立请求方式为例，来自中断请求寄存器 IR 的多个中断请求信号与中断屏蔽寄存器 INM 逻辑与后进入优先编码器，如图 1.26 所示，优先编码器根据响应优先级生成中断请求的编号 xy，完成中断识别，所有 IR 信号逻辑或后得到中断请求信号与中断使能寄存器 IE（开中断，关中断指令控制）相与后送 CPU，CPU 执行完一条指令后如果发现有中断信号则进入中断处理流程。

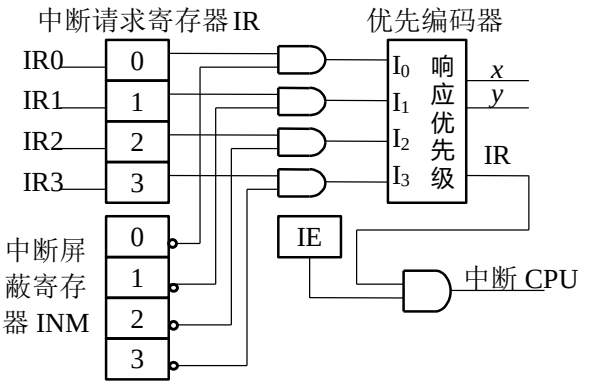

图 1.26 中断屏蔽与中断识别

在获知具体中断源后，还需要获得中断服务程序入口地址，这样才能执行中断服务程序。获得中断服务程序的入口地址的方法主要有两种：向量中断法和非向量中断法。其中向量中断法对应独立式中断请求方式，**向量中断法**将中断向量（中断服务程序的入口地址和程序状态字 PSW）以数组形式组成一个**中断向量表**，用中断类型编号（中断号）作为小标进行检索，类似中断入口地址查找表。

向量中断法中断响应流程如下：先将各个中断服务程序的中断向量组织成中断向量表；中断响应时，通过相应方法识别中断源获得中断号，然后通过中断向量表入口地址计算得到该中断的具体中断向量地址，再根据向量地址访问中断向量表，从中读出中断服务程序的入口地址，并装入程序计数器 PC 中和条件状态寄存器中，CPU 开始执行中断服务程序。

当一个中断号被多个同类设备共享时（链式，或二维链式结构），引起该中断的具体设备必须采用非向量中断法获取具体中断服务程序入口地址。非向量中断法的中断响应方式为：CPU 在响应中断请求时，只产生一个固定的地址，该地址是中断查询程序的入口地址，通过执行该查询程序来确定中断服务程序的入口地址，然后执行响应的中断服务程序。

(6) 中断时机与中断响应

CPU 响应中断的时机是指令执行周期结束后，判断中断请求信号 IR 是否为高电平，如为高电平则暂停程序执行，进入中断周期，首先通过将中断使能寄存器 IE 清零的方式关中断，并保存程序断点---指令寄存器 PC 的值，MIPS 中是将 PC 送 EPC，同时进行中断源识别，获取中断入口地址，并将中断入口地址送 PC，这部分全部由硬件逻辑完成，需要一定的时间开销（可能需要若干时钟周期，此阶段 CPU 并不能执行指令），相关动作实际也对应一系列数据通路，也将这部分动作称为中断隐指令（实际它并不是 ISA 指令集的一部分），如图 1.27 所

示。中断服务程序入口地址送 PC 后，CPU 即完成了暂停现有程序，转去中断服务程序的过程。

值得注意的是，缺页中断是一个例外，由于发生缺页中断时指令并不能执行完毕，所以缺页中断的时机并不是指令执行周期结束后的公操作，而且这条未能执行的指令中断结束后应该重新执行，所以其保存的断点 PC 的值也不一样。

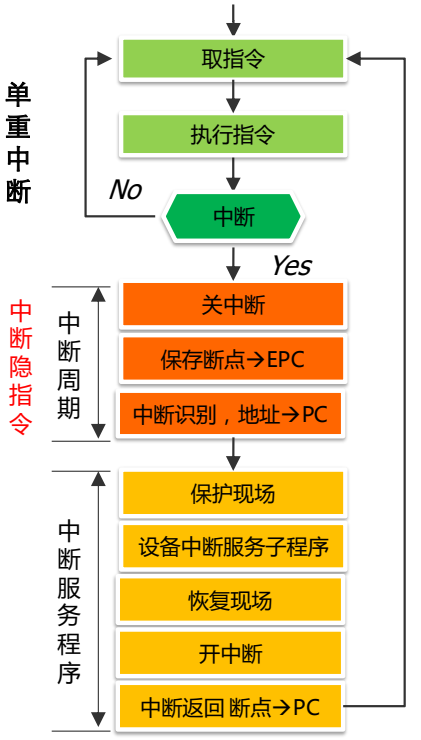

图 1.27 单级中断流程

(7) 中断服务

单级中断系统中中断服务程序首先是通过堆栈压栈的方式保护现场，现场包括所有可能被破坏的通用寄存器，状态字等，与普通函数调用中的保护现场不同，中断服务程序并没有调用者，所以本应该由调用者函数负责保存的部分寄存器也需要中断服务程序保存。保护现场完成后正式进入为设备服务的中断服务子程序，进行设备数据交互，该程序结束后通过堆栈出栈的方式恢复现场，然后开中断，允许 CPU 响应其他中断，最后进行中断返回，其任务是将断点地址送 PC。值得注意的是，**开中断应该与中断返回同时完成**，否则如果在中断返回指令执行之前有中断请求，断点将被破坏。

整个中断处理流程中中断隐指令部分是由硬件实现，中断服务部分是由 CPU 执行中断服务子程序实现，其中包含保护现场恢复现场等较为复杂的访存操作，这些都需要占用 CPU 时间，一次中断过程只能传输一个字，所以中断机制比较适合为外设的随机事件服务，不太适合高速的数据传输。

1.6.3 实验内容

为已经实现的单周期 MIPS CPU 增加单级中断处理机制，引入 3 个外部中断源，如图 1.28 左下角所示，该 CPU 支持 3 个 Logisim 按钮触发的中断源，分别对应中断请求 IR1，IR2，IR3，3 个 LED 指示灯 W1,W2,W3 表示当前中断事件正在等待响应，3 个中断源对应不同的中断优先级，其中 3>2>1，要求按下按键 1 后能进入数字 1 的走马灯演示子程序，按下按键 2 后进入数字 2 的走马灯演示子程序，按下按键 3 后进入数字 3 的走马灯演示子程序。

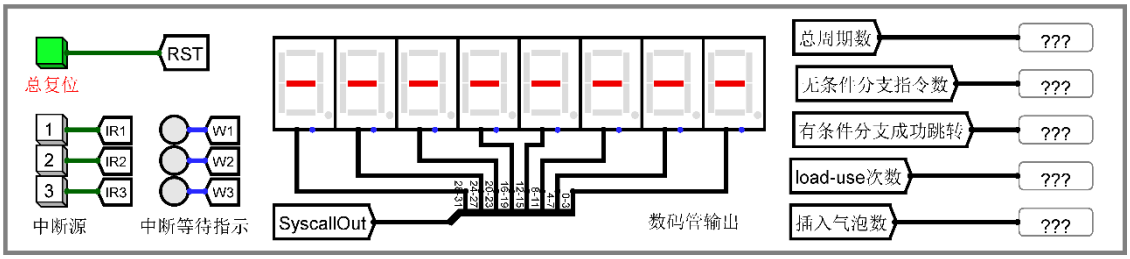

图 1.28 支持单级中断的单周期 MIPS CPU 输入输出引脚

1.6.4 实验步骤

（1） **研究 MIPS 中断处理机理**。MIPS CPU 中断控制一般通过协处理器 CP0 完成，CP0 是独立于通用寄存器文件的另外一组寄存器，这些寄存器不能直接被 MIPS 指令集引用，MIPS 指令集中由两条专用指令负责 CP0 通用寄存器的读写（MFC0，MTC0），中断使能位 IE，中断屏蔽位均在 CP0 通用寄存器组中有详细约定，但其实现较为复杂，为简化实验，我们可以不遵循 MIPS CP0 的约定，按照本实验的需求进行最大的简化，能实现中断机制即可。

（2） **设计中断按键信号产生电路**，具体可参考如图 1.29 所示，其中同步清零信号用于清除中断请求信号。

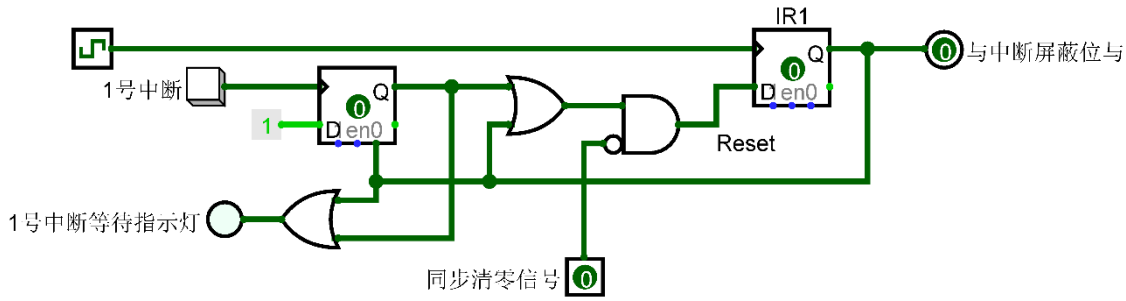

图 1.29 中断按键采样电路

（3） **设计中断识别机制**，能实现实验要求中断响应优先级，可参考图 1.26，能识别中断源

编号，并设计由中断号寻找中断程序入口地址的逻辑电路。

（4） **存储区间规划**，在系统中要考虑数据内存区间如何划分，数据区和堆栈区如何分配，堆栈区主要用于中断服务程序中的保护现场和恢复现场的堆栈操作，需要考虑堆栈指针$sp 如何进行初始化，放在哪个地址合适，需要考虑指令存储器的区间规划，中断服务子程序放在那里合适？

（5） **设计中断隐指令数据通路**，保存断点至 EPC、中断识别，中断服务程序入口地址送 PC 等，可参考图 1.27，压栈涉及的访存操作，中断识别寻找入口地址有可能涉及访存操作，可能需要多个时钟周期，这个过程中 PC，以及其他功能部件如 ALU，REGFILE 等如何动作？

（6） **增加 mfc0,mtc0，eret 指令的支持**，需要重新设计控制器。

（7） **编写中断服务子程序**。

（8） **系统联调**，功能测试。

1.6.5 实验思考

(1) 函数调用 JAL 与中断有多大区别，有否共同之处？

JAL 与中断的处理过程及实现的功能类似，都是将 PC 保存到某个寄存器，然后修改 PC 的值。不同的地方在于：JAL 将 PC 保存到 31 号通用寄存器，中断则将其保存到另外一个特殊的寄存器 EPC 中；修改后 PC 的新值，在 JAL 中由指令给出，中断则直接由硬件确定；JAL 是一条指令，用户在程序中显示调用，当 PC 指向该指令时执行，中断则不是一条指令，它是在满足中断响应条件时执行（外部中断信号有效，中断使能，且中断没有被屏蔽）。

另外二者进入具体函数时都需要保存现场，这里的现场就是被调用者保存寄存器，一般不同指令系统会事先约定好，便于编译器生成代码，但中断保护的现场还应该包括被破坏的调用者保存寄存器，因为中断程序并没有调用者函数。

(2) 不同的中断请求存储在哪里，何时消失？

不同的中断通常保存在一个特殊的中断请求寄存器 IR 中，其寄存器的每 1 位对应一个中断请求。对应中断请求被中断响应处理时应该消失，这里应该由硬件完成这一清零动作。

(3) 硬件响应优先级用什么电路实现，为什么要有处理优先级？

硬件响应优先级一般用优先级编码电路来实现，这样优先级高的中断请求就会最先被 CPU 响应。只有没有更高的中断请求时，电路才会把较低优先级的中断请求送给 CPU，优先编码器电路在 logisim 中有现成的模块。

(4) 中断使能寄存器 IE 是干什么用的？

中断使能寄存器就像是一个开关，用来打开或关闭 CPU 的中断请求，外部设备的中断请求信号会与这个开关进行逻辑与操作。所以中断使能有效时，CPU 才能响应中断请求，无效时 CPU 不会响应任何中断请求，软件中开中断，关中断指令，如 X86 指令集中的 STI、CLI 指令就是

控制这个开关的，MIPS 中必须利用 CP0 寄存器组访问指令来控制对应的位，具体位在 MIPS 中有约定，CP0 寄存器中的某一位实现了 IE 的功能，但实验实现的时可以根据需要灵活调整。

MIPS CPU 则可以通过下面的方式来修改 IE 寄存器，从而实现开中断、关中断处理：用 MFC0 指令将 IE 的值放到一个通用寄存器，修改这个通用寄存器（根据需要置 0 或 1），然后用 MTC0 指令将这个值从通用寄存器写回 IE 寄存器，从而完成 IE 的控制。

(5) CPU 如何判断当前有中断需要响应？

只有对应中断请求有效，同时该中断请求没有被屏蔽，且中断使能有效，CPU 才能收到最终的中断请求信号，当当前指令执行完毕时，CPU 才会进行中断响应。

(6) CPU 发现当前存在中断事件后要做什么动作，什么时候响应中断事件？哪些是硬件完成，哪些是软件完成？由硬件完成的动作需要多少个时钟周期，此时 CPU 能否执行指令？

CPU 发现当前有中断事件后，在可以响应中断时要完成下面这些动作：（1）保存断点，打断的指令的 PC 值；（2）中断识别并跳转，根据中断源找到中断程序入口 地址，修改新的 PC 值转到中断处理程序；（3）完成中断服务的准备工作；（4）执 行中断服务程序；（5）结束中断服务，完成中断返回的准备工作；（6）中断返回。 其中（1）和（2）由硬件完成，也就是所谓中断隐指令干的事情，这部分需要一个或者多个时钟周期，具体与实现有关，其它则是软件实现。 由于硬件需要完成（1）和（2）二步工作，可以 1 个时钟周期完成，也可以用 2 个时钟周期来完成，这个过程中 CPU 其实是相当于在执行指令（见问题 1，类似于 JAL 指令）因为指令的执行也就是建立相应的数据通路，因此此时 CPU 不能执行其它指令。

(7) 单级中断断点保存在哪里？

单级中断的断点通常保存在一个专用的寄存器中，就 MIPS CPU 而言，是保存在 CP0 协处理器中的一个叫做 EPC 的寄存器中。

(8) 中断处理程序中的现场有哪些，我们实验中需要考虑保存哪些现场？

理论上中断处理时 CPU 要转去执行另外一个程序，执行完后返回再继续执行被打断的程序，因此将改变 CPU 状态的任何操作之前，要被改变的状态都需要保存起来，返回再恢复。CPU 的状态都是通过寄存器实现的，所以所谓保护现场就是要备份即将修改的寄存器的值。我们实验中需要考虑保存的现场主要有：中断返回地址 EPC、中断屏蔽字、以及在中断服务程序中将使用的通用寄存器。记住一个原则，你将修改什么，在修改之前最好就保存什么。

(9) 中断程序入口地址如何识别？

本实验中有 3 个中断源，资源不受限，最简单的方法就是采用独立式中断请求进行连接，利用优先编码器电路即可实现中断源的识别，再增加中断向量表逻辑实现中断号到中断入口地址的转换，但中断向量表涉及内存数据访问，逻辑较为复杂，为简化实验设计，中断向量表可以用组合逻辑电路固化。

(10) 开中断，关中断在 MIPS 指令集中如何实现？

MIPS 处理器没有专门的开中断、关中断指令，具体实现时时利用利用 CP0 寄存器组访问指令 MFC0,MTC0 实现的

(11) 中断处理程序放在指令存储器中的那个位置，如何载入到 ROM 中？

中断处理程序放在指令存储器的一个特定位置，它的起始地址必须与中断响应时 CPU 赋予 PC 的值相同，按照习惯中断处理程序的入口地址通常是存储器最低的地址，即 0x00000000。中断处理程序一般是与主程序一起编译，然后统一做成存储器文件 用原来相同的方式载入 ROM。（放哪应该无所谓，实验中自己安排，放低地址 CPU 上电 PC 初始值要改）

(12) 数据堆栈放在哪里？SP 寄存器如何设置？MIPS 如何访问堆栈？

数据堆栈通常放在数据存储器中。MIPS 处理器的堆栈是向下生长的，因此 SP 寄存器 初始指向存储器的最高地址，每保存一个值，SP 要减 4；出栈时则 SP 加 4。例如：

```
# 设置 SP 指针
li sp, STACK_BASE_ADDR

# 入栈
addiu   sp, sp, -4
sw  ra, 0(sp)

# 出栈
lw  ra, 0(sp) addiu sp, sp, 4
```

(13) 按键中断是电平触发还是跳变触发？连续按键如何处理？实际系统中是如何处理的？

按键一般是边沿触发，但是处理边沿中断触发通常较为困难，因此常用锁存器将边沿触发转换为电平信号，然后用电平中断触发。连续按键最简单的处理办法是在第一次按键没有处理完之前，不管连续的按键信息，即发生按键时锁存器保存，然后这个锁存器就锁住，不再保存新的按键信息，只有当 CPU 把这个按键中断处理完毕后，再开放这个锁存器。实际系统中这个锁存器是一个缓存，可以保存按键的队列信息，这样 CPU 就可以一个一个的处理，直至保存的按键缓存队列全部被处理。（实际系统运行是中断处理非常快，操作系统也规定中断服务程序不能太长，否则系统会 down 掉， 连续按键都可以得到及时响应）

(14) 实验中的中断机制为啥要用 CP0，不要是否可以？在我们的实验中如何简化？

之所以用 CP0 是为了方便程序的编译（因为要用标准的 MARS 工具来编译程序，因此符合 MIPS 规范的实现方式就比较方便）。其实是可以简化的，另外再增加 1-2 个 寄存器就可以解决问题。开中断关中断等与中断相关的操作必须访问 32 个通用寄存器以外的寄存器，

而通用指令中只有 5 位表示寄存器，所以无法访问与中断相关的寄存器，所以必须借助 CP0 寄存器访问指令来实现中断相关的寄存器访问。实验中 CP0 寄存器可以不遵守 MIPS 规范，只要能实现中断相关机制即可。

1.7 多级中断实验

1.7.1 实验目的

学生掌握多级嵌套中断处理机制，能在单周期 CPU 中设计多级嵌套中断处理机制，能处理多个外部中断事件，能正确的终止主程序的执行，转为为按钮事件服务的中断服务程序，中断服务程序执行完毕后应返回主程序继续运行，不同的按钮会进入不同的中断服务程序，高优先级中断可以打断低优先级中断，高优先级中断服务器程序执行完毕后应能返回被中断的中断服务程序，直至主程序。

1.7.2 背景知识

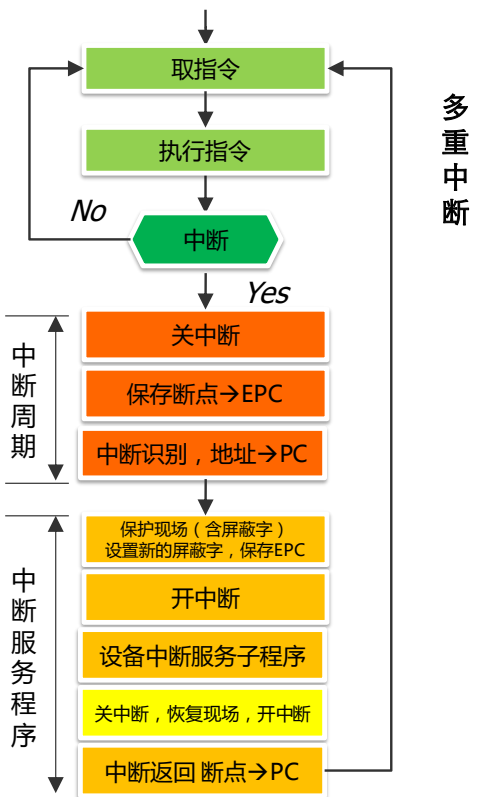

图 1.30 多级嵌套中断流程

多级中断与单级中断的区别是中断可以嵌套，其在中断隐指令阶段的逻辑与单级中断完全一致，区别在于中断服务程序流程略有区别，具体处理流程如 XXX 所示，中断服务程序逻辑部分保护现场应该包括中断屏蔽字的保护，方便动态调整中断优先级，另外由于中断程序需要实现嵌套，所以 EPC 可能被破坏，所以保护现场过程中必须保存 EPC，现场保护结束后立即开中断以使得系统可以接受新的中断，中断服务子程序结束后，要进行恢复现场的操作，注意恢复现场应该是原子操作，必须在关中断的前提下进行，注意开中断和中断返回也应该同时进行。

1.7.3 实验内容

为已经实现的单周期 MIPS CPU 增加多级中断处理机制，引入 3 个外部中断源，如图 1.28 左下角所示，该 CPU 支持 3 个 Logisim 按钮触发的中断源，分别对应中断请求 IR1，IR2，IR3，3 个 LED 指示灯 W1,W2,W3 表示当前中断事件正在等待响应，3 个中断源对应不同的中断优先级，其中 3>2>1，要求按下按键 1 后能进入数字 1 的走马灯演示子程序，按下按键 2 后进入数字 2 的走马灯演示子程序，按下按键 3 后进入数字 3 的走马灯演示子程序，**高优先级中断能正确中断低优先级中断服务子程序**。

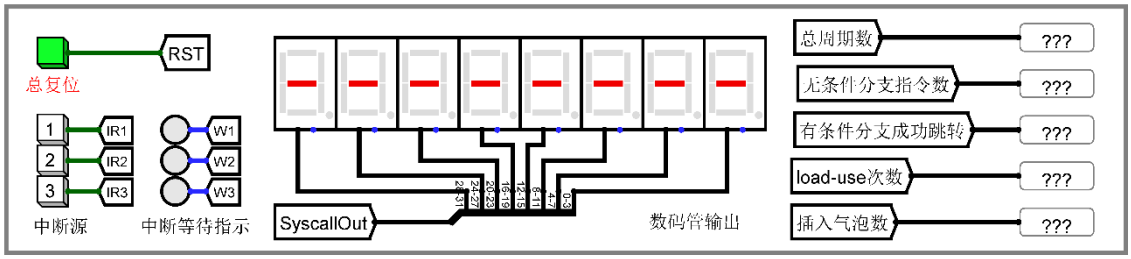

图 1.31 支持多级嵌套中断的单周期 MIPS CPU 输入输出引脚

1.7.4 实验思考

(1) 多级嵌套中断的断点如何处理？

单级中断的断点保存在 CP0 协处理器中的一个叫做 EPC 的寄存器中，但 EPC 只是一个寄存器，进行多级中断嵌套时，会被后面的中断破坏，所以中断服务程序中应该讲 EPC 作为现场保存起来，EPC 实际上也是一个被调用者保存寄存器，类似 JAL 嵌套调用，需要保护$ra 寄存器一样。

(2) 高优先级中断服务程序执行过程中，有新的按键事件发生，如何处理？

由硬件判断新的按键的优先级的高低，如果比当前中断服务程序的级别高，就打断当前的中断服务程序（类似于被打断的主程序），转而执行这个更高优先级的中断服务程序；如果比当前的中断服务程序级别低，则需要等待当前的中断服务程序执行完毕并返回后，才能够进行中断响应；如果是与当前的中断服务程序级别一样，也类似低优先级中断，可以像上面一样类

似处理。注意 CPU 响应不响应中断取决与能否收到中断请求信号，如果收不到就不管。

(3) 中断屏蔽寄存器有什么作用，何时设置中断屏蔽字，真实计算机环境中由什么程序设置中断屏蔽字？本实验是否需要中断屏蔽寄存器？

中断屏蔽寄存器的作用是保存中断屏蔽字，中断屏蔽字的用途是动态的调整中断处理优先级；正常情况下，优先级高的中断请求在被处理时， 即当它的中断服务程序在执行时，是不允许被它同级或更低级别的中断请求打断的。，但是通过设置中断屏蔽字，可以允许它被低级别的中断请求打断，从而先完成低级别中断请求的中断服务程序，然后再执行完它自己的中断服务程序。

中断屏蔽字是在中断处理过程中设置的，在真实计算机环境下，通常是由中断服务程序进行设置。由于本实验在高级别中断请求在处理时不允许被低级别的中断请求打断，因此其实不需要中断屏蔽寄存器。

1.8 流水中断实验

1.8.1 实验目的

学生应用流水相关知识，中断相关知识，为流水 CPU 增加中断处理机制，最终能正常运行并能正确运行标准测试程序，并能演示 3 个中断源的中断。

1.8.2 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。

1.8.3 实验思考

(1) 单周期 CPU 中断处理和流水中断处理有何区别？

在单周期 CPU 中，每指令执行完毕后，CPU 就可以去判断是否有中断需要处理， 如果有就进行中断处理，如果没有则正常执行下一条指令。但是在流水线中就较为复杂，因为在流水线中同时有 5 条指令在执行，这 5 级指令还分别处于不同的处理 阶段，即 IF、ID、EXE、MEM、WB 各有一条指令在执行，另外甚至同时有多条指令都进入了结束阶段，中断哪条指令，同时执行的其它指令怎么办 都需要考虑。

理论上，5 个流水段的任意一个段上都可以处理中断，为了实现的方便通常会选择一个段来处理中断，可供选择的段常常是 EX 或者是 WB。但是，不管选择那个流水段来处理，都要做下面的事情（以在 EX 段处理中断为例，其实在 WB 段处理更加方便）：（1）EX 段前面的 2

条指令要继续执行完；（2）EX 段后面进入流水线的 2 条指令要取消；（3）处于 EX 段本身的这条指令有 2 种选择，如果这条指令允许继续执行完，返回的断点地址应该是该指令的 PC+4；如果不允许这条指令继续执行，返回的断点就是这条指令的 PC；（通常做法是不允许这条指令继续执行）；另外如有指令在 EX 段之前完成，比如无条件分支指令在 ID 段完成，此时逻辑又有区别。（4）暂停流水线进行中断响应；（根据具体的实现方法也可以不暂停）（5）重新启动流水线从新的 PC 值处取指令（新的 PC 即是中断入口地址）。

1.9 动态分支预测实验

1.9.1 实验目的

学生研究动态分支预测相关原理，掌握相联存储器设计机理，并最终应用相关机制为已经实现的五段流水 CPU 添加动态分支预测机制，最终方案应比原有方案性能更优。

1.9.2 背景知识

采用重定向机制后，指令流水线中数据相关基本不需要插入气泡就可解决，只有少数 Load-Use 冲突需要插入一个气泡解决冲突问题，流水线性能得到较好的提升。此时指令流水线中结构冲突（分支冒险）对流水线性能影响很大，分支指令会使得流水暂停，误取指令被清空，引起分支延迟，指令误取深度越长，对指令流水性能影响越大，所以应尽量提前无条件跳转指令的执行段，但最多在 ID 段完成无条件跳转的执行，这是减少指令误取深度的方法。

另外为进一步减少分支带来的延迟，应在流水线中今早判断出分支是否成功跳转，应该尽早计算出分支目标地址，目前减少分支延迟主要方法有静态分支处理与动态分支预测两种方法，静态分支预测主要是基于编译器的编译信息对分支指令进行预测，预测信息不再改变。动态分支预测是依据程序运行时的实时信息，不对的对预测信息进行更新，具有较高的预测准确率，需要硬件支持。其中静态分支处理主要是采用一些简单的策略进行预测或处理，主要有如下基本策略：

1) 预测分支失败，这是缺省逻辑，与实验中的方案相同。
2) 预测分支成功，这个逻辑也不会比前面的方案好很多。
3) 延迟分支，ID 段发现当前指令是分支指令时，IF 阶段继续取一条指令，作为分支指令的延迟槽 delay slot，这条指令不管是否跳转，都会被执行，这种思路也会减少误取深度，提升流水线性能，这需要编译器的支持，在本实验中没有考虑延迟槽的概念。

所谓动态分支处理方法，是指利用程序执行期间的历史统计信息进行分支预测处理，为提升性能，历史统计信息的逻辑必须用硬件实现。现代处理器中均支持动态分支预测算法，

静态分支预测中不管预测成功还是失败都是恒定的逻辑，无法利用历史信息，动态分支预测方法可以利用历史信息进行预测，提升预测准确率，最简单的动态分支预测策略是分支预测分支历史表 BHT(branch history table)，有时也称为分支预测缓冲器（Branch Prediction Buff)

或，BHT 表中存放跳转指令的地址，以及过去的跳转信息描述位（预测位），研究表明，双预测位即可实现较高的预测率，采用两位分支预测位的 BHT 状态转换图如图 1.32 所示。当状态位位 00,01 时预测跳转，位 10 和 11 时预测不跳转，当前指令实际分支成功与否决定状态的变迁。

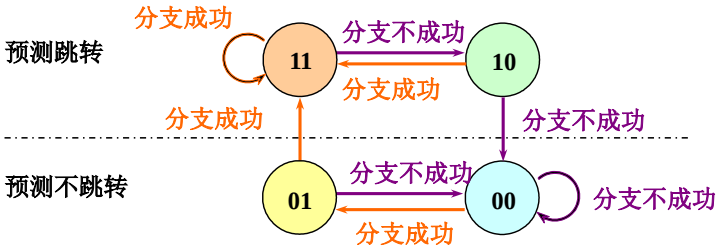

图 1.32 采用双位预测位的 BHT 状态转换

ID 段译码发现是分支指令后，可以根据 BHT 表查找当前分支指令的分支预测位，根据分支预测位的值预测跳转还是不跳转，如果跳转，需要同时计算分支地址送 PC，这种方式可以提升预测准确率，及时发现分支指令并给出正确的分支地址，但预测正确时还是存在一个误取指令，如果能在 IF 段就能进行预测才能将分支延迟降低到 0，分支目标缓冲器 BTB（Branch Target Buffer）就可以解决这个问题，BTB 表存放曾经成功跳转的分支指令的 PC 和分支目标地址，可以看成一个分支目标地址 Cache。

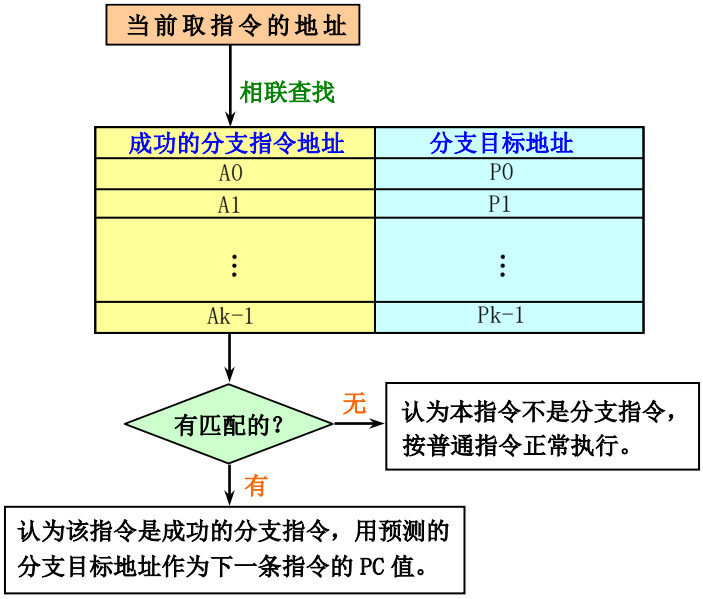

图 1.33 BTB 的结构

BTB 逻辑结构如图 1.33 所示，BTB 中存储的表项是（成功分支的指令地址 PC，分支目标地址），取指令时利用当前 PC 地址与该表格做全相联并发比较，如果发现有相符的说明该指令上次成功跳转，直接用分支目标指令进行下条指令的寻址，如果没有匹配的，说明这条指令

不是分支指令或者上次并未分支跳转成功，直接按 PC+4 的方式取下一条指令。具体流程如图 1.34 所示：

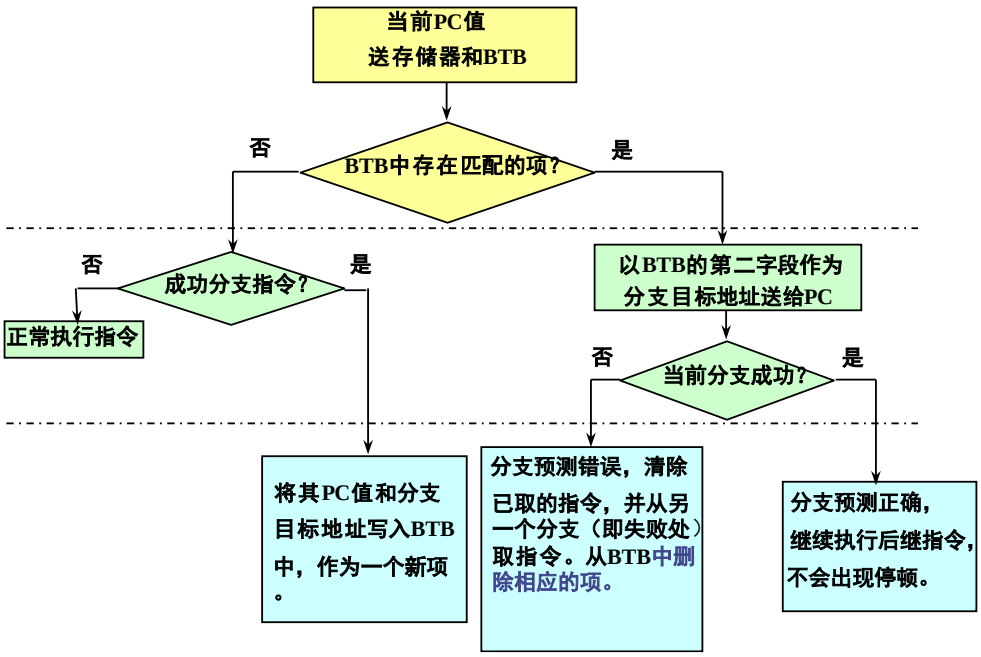

图 1.34 采用 BTB 时的初步流程

这里 BTB 实际是一个相联存储器，利用 PC 作为关键字进行数据访问，输出是分支目标地址。

1.9.3 实验内容

为流水 CPU 增加动态分支预测逻辑，取指令阶段可以以 PC 的值为关键字查询 BHT 表（Branch History Table），根据历史统计信息直接预测下条指令的正确地址，通过提升预测准确率，尽量避免分支指令引起的流水线清空暂停现象，从而优化流水线性能。

要求将 BTB 表和 BHT 表进行融合在 IF 段进行分支预测，BTB 每个表项内容应包括 Valid 位、PC、BranchAddr、双预测位、LRU 调度标记(增加了双预测位)。实际处理流程取指令阶段以 PC 为关键字到 BHT 表做全相联比较查找，如果命中，直接根据预测位决定是否跳转，并给出下一条指令正确地址（取指令阶段，与取指令并行）。分支指令执行阶段会根据跳转与否更新 BTB 中的表项，如当前执行指令不在 BTB 中，需要进行淘汰，如已经存在，根据状态机更新双预测位的值，同时将 LRU 调度信息清零，以提升下次预测的准确性，具体流程如图 1.35 所示。

1.9.4 实验步骤

(1) 设计 BTB 表，实现全相联机制，实现 LRU 淘汰算法。

(2) 修改分支指令执行逻辑，跳转成功失败信息需送 BHT 表进行统计，以方便决策？新的跳转指令应载入 BHT 表，如 BHT 表已满，需要进行淘汰。

(3) 系统联调，功能测试。

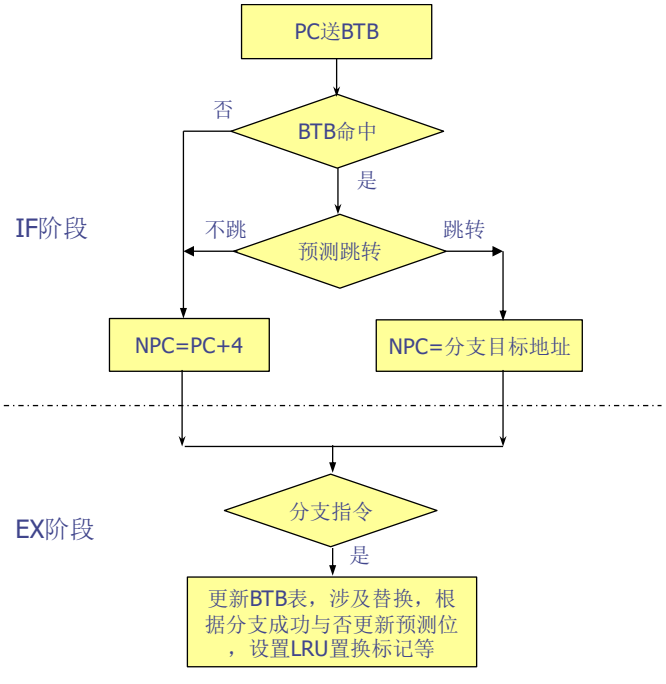

图 1.35 动态分支预测流程流程