**计算机体系结构实验课程第 三 次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 多周期CPU实现 | | | 班级 | 李雨森老师 |
| 学生姓名 | 蒋薇 | 学号 | 2110957 | 指导老师 | 董前琨 |
| 实验地点 | A308 | | 实验时间 | 2023.10.23 | |

1. **实验目的**

1. 加深对计算机组成原理和体系结构理论知识的理解。

2. 培养对 CPU 设计的兴趣，在理解现有 CPU 架构的基础上，引发对体系结构的思

考和创新。

3. 培养创新思维能力，并通过实践验证新想法。

1. **实验内容说明**

1. 分析静态 5 级流水 CPU 中的流水线阻塞情况，包括数据相关、控制相关、结构

相关等，优化流水线设计，尽可能减少流水线阻塞情况，比如前递技术等。

2. 对于分支跳转指令，mips 架构中有延迟槽指令的设定，利用这一点，在静态 5

级流水 CPU 中，可实现分支指令永远不阻塞后续指令，大家可以检查自己的流水线设

定，进行优化实现这一点。

3. 针对第 2 点，此时，分支跳转指令就不需要进行转移猜测了，但大家可以将流

水线结构改为 x86 中无延迟槽技术的设定，此时分支跳转指令与后续真正需要执行的

指令至少会堵塞一拍，此时可以考虑实现转移预测技术，提升流水线结构。

4. 在学习的过程中，大家一定会有很多自己的想法，比如，为什么取指、译码、

执行、访存、写回称为经典的 5 级流水结构，可以实现 3 级、4 级、6 级流水结构的

CPU 吗?答案肯定是可以的，甚至在各类产品的 CPU 中采用经典 5 级流水结构的都很

少。所以希望大家尽情地发挥自己的想法，大改流水结构，验证自己关于流水结构的

想法，您可以实现 3 级（比如：取指、译码、执行）、4 级、6 级等等各类流水结构的CPU。

5. 课程设计要求大家实现的 mips 指令有限，大家可以分析其余 mips 指令，加以

实现。

6. 目前课程设计实现的指令存储器和数据存储器是同步读的机制的，故在当前拍

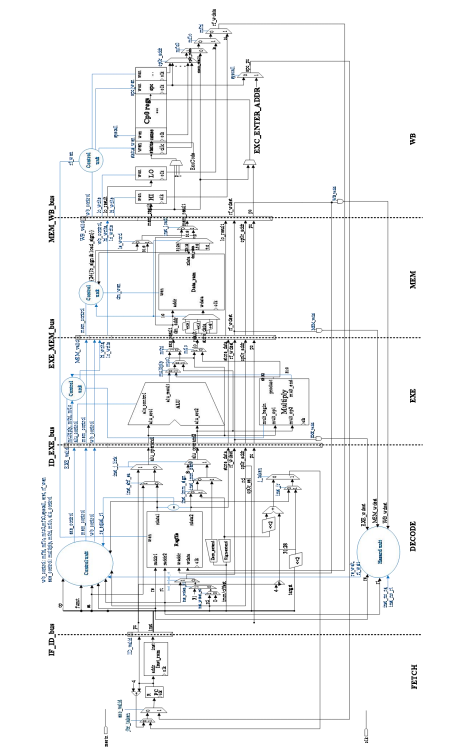
数（时钟周期）发出读数据的地址请求时，在下一拍才能获得读的数据，因此取值级

和访存级的 load 都需要两拍时间。其实发地址请求在下一拍获得读的数据，明显也是

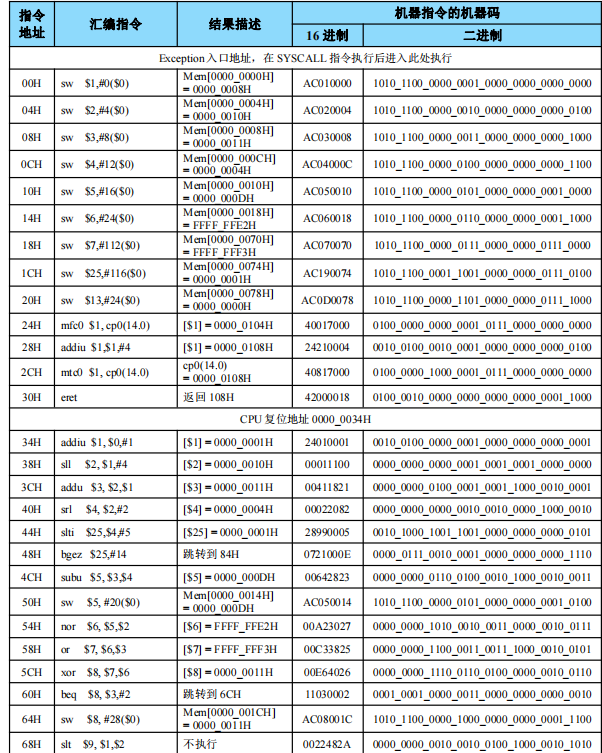
一个可以流水做的工作，故可以考虑对流水线设计方案作稍微修改，使得取指级和访

存级的 load 不需要多等一拍。

**五级流水线CPU 的实现框图:**



**五流水线 CPU 测试所用汇编程序：**



1. **实验步骤**
2. 使用前递技术减少阻塞情况

**指令相关和流水线冲突**

目前流水线结构把一条指令的来去过程分成5个阶段，分别是取指（IF）、译码（ID）、执行（EX）、访存（MEM）和写回（WB）。计算PC和译码阶段是并行的，可以视作译码的一部分。虽然这样的流水可以提高主频，但由于执行间的相关可能会导致执行结果的错误。如第N条指令把结果写到R1寄存器，第N+1条指令要用到R1的值进行运算，由于第N+1条指令读R1发生在译码阶段，此时R1的值还没更新，从而发生错误。

写后写相关（WAW）

**定义**：即两个指令写同一个单元，在乱序执行的结构中可能会发生此类错误，静态流水线天然保序，不会出现这种问题

读后写相关（WAR）

**定义**：在乱序执行的结构或者读写指令流水级不一样时，如果后面的写指令执行得快，在前面的读指令读书之前就把目标单元原来的值覆盖掉了，导致读书指令读到了该单元“未来”的值，从而引起错误

**控制相关**

冲突的本质：对程序计数器PC的冲突访问引起的

**解释**：在上图中第一条指令的ID到第二条指令的IF之间的箭头表示控制相关引起的冲突，即如果第一条是转移指令，第二条不是延迟槽指令的话，则第二条指令等一拍。这里要强调的是，NPC总是需要计算才能得到，将PC相关的计算部件放在了ID阶段而非EX阶段，就可以仅打一拍完成转移指令的后续操作，否则等写回时才拿到NPC的结果就要等4拍，从而造成大量的空泡。

**数据相关的联动**：如果转移指令在译码阶段根据寄存器的值进行条件判断时，该寄存器是EX、MEM和WB阶段的目标寄存器时，即转移指令与先前的指令存在数据相关，转移指令也会由于数据相关堵在ID流水级，延迟槽指令则堵在IF流水级

**解决方法**

加延迟槽这个解决控制相关的方法仅适应于单条静态流水线，而现代处理器的超流水、多发射、乱序执行流水线中，则需要复杂的转移猜测技术

**结构相关**

问题举例

取指令和访存有单独的存储器，如果共用一个的话，取指和取数都访存时就会冲突，方寸的时候下一条指令就不能取指了。

由于除法不能全流水的实现，前一条除法指令在运算时后面的触发指令需要等待

5级流水中，尽管普通运算指令不用访存，但也要经历MEM流水，本质原因是如果有写指令5拍写回，而有的4拍写回，就会引起通用寄存器堆写端口的结构冲突

**流水线的前递技术**

5级静态流水线通过指令的阻塞来保序，但阻塞必然导致流水线效率的降低，可以通过软硬件的方法提高流水线效率。

**前递技术（Forward），又称旁路技术（Bypass）**

**原理**：若前后的指令有数据相关，将前面的指令直接把运算结果传给后边的指令而非等到写回寄存器后再给后边的指令结果。

**具体实现：在流水线的运算器前通过多路选择直接把前面指令的运算输出作为后面指令的输入。** 带有前递技术的流水线下图是在原来流水线的基础上添加了部分前递通路的情况，且只考虑数据前递给ALU，不考虑前递给存数指令和转移指令的旁路。

ALU的每个输入端都添加一个3选1逻辑，3个输入分别是原来的ALU输入、下一级流水线输出的结果（即EX流水级ALU的运算结果）和再下一级流水线输出的结果（即MEM流水级的结果)。**这样后面指令要用到前面指令的运算或访存结果时，就可以直接通过运算器前面的多路选择器选择前面指令的运算或访存结果，不用等到前面指令把结果写回到寄存器后再从寄存器中读取**

**前递的判断**：需要比较处于EX阶段的指令的源寄存器号和处于MEM或者WB阶段的指令的目标寄存器号是否相等，如果相等且不是0号寄存器，则说明处于EX流水级的指令与前面的指令有数据相关，需要直接读取前面指令的结果用于运算器的输入。

硬件开销：采用前递技术后就要把指令的两个源寄存器号传递到EX流水级，分别是图中的src1E和src2E，

1、从执行阶段（EX）前递到译码阶段（ID）：当一个指令在执行阶段计算出结果后，该结果可以直接传递给后续的指令进行译码和操作数读取;2、从访存阶段（MEM）前递到译码阶段（ID）：当一个指令在访存阶段计算出结果后，该结果可以直接传递给后续的指令进行译码和操作数读取;3、从执行阶段（EX）前递到访存阶段（MEM）：当一个指令在执行阶段计算出结果后，该结果可以直接传递给后续的指令进行访存操作。

1. 针对分支跳转指令，实现不阻塞后续指令执行:

在MIPS架构中，分支跳转指令会引入分支延迟（branch delay），即指令流水线在分支指令后的一个周期会执行分支指令后的指令而不是下一条指令。这是因为分支指令的条件判断需要在取指阶段完成，而取指阶段之后的指令已经进入流水线。

**延迟槽**：将分支指令后面的一个指令作为延迟槽指令，将其与分支指令一起执行。这样，在分支指令的条件判断结果出来之前，延迟槽指令可以继续执行，避免了分支指令造成的流水线停顿。

延迟槽是指在分支指令后面的一个位置留出的空槽，可以放置一个指令，这个指令会在分支指令判断结果之前执行。

延迟槽中的指令应该是不依赖于分支结果的指令，因为在分支指令条件判断之前，延迟槽中的指令已经开始执行了。如果延迟槽中的指令依赖于分支结果，可能会导致错误的结果。

1. 针对分支调整指令，考虑加入分支预测，提升流水线性能:

一位：定义一个bool类型的变量branch\_prediction来表示分支预测结果，在update\_branch\_prediction函数中，我们将分支预测的结果存储在branch\_prediction变量中。

1. #include <stdbool.h>
2. *// 分支预测*
3. bool branch\_prediction = false;
4. *// 分支预测器更新*
5. void update\_branch\_prediction(bool prediction) {
6. branch\_prediction = prediction;
7. }
8. *// 分支调整指令*
9. void branch\_adjust(int address) {
10. *// 分支预测*
11. if (branch\_prediction) {
12. *// 执行分支跳转的操作*
13. *// ...*
14. } else {
15. *// 执行分支不跳转的操作*
16. *// ...*
17. }
19. *// 更新分支预测结果*
20. *// 根据实际情况更新分支预测器*
21. update\_branch\_prediction(is\_branch\_taken);
22. }

在branch\_adjust函数中，我们首先使用branch\_prediction变量进行分支预测，根据预测结果执行相应的操作。然后，我们通过调用update\_branch\_prediction函数来更新分支预测结果。

两位：转移预测技术是一种通过硬件来预测分支指令的跳转方向的方法，以尽可能减少流水线停顿和分支指令带来的性能损失，动态转移预测：通过硬件中的转移预测缓冲器（Branch Prediction Buffer）来记录分支指令的历史执行信息和模式，并根据这些信息预测下一次的跳转方向，二位饱和计数器（Two-bit Saturating Counter）：使用一个2位的计数器来记录分支指令的历史执行情况。根据计数器的值，可以预测分支指令的跳转方向。

使用一个branch\_counter\_t枚举类型的变量branch\_prediction来表示分支预测结果。通过2位计数器的方式，我们将分支预测结果分为4个状态：STRONGLY\_NOT\_TAKEN、WEAKLY\_NOT\_TAKEN、WEAKLY\_TAKEN和STRONGLY\_TAKEN。

在update\_branch\_prediction函数中，我们根据实际的分支执行情况更新分支预测器的状态。如果分支被正确预测，则将计数器的值向预测方向移动一位；如果分支没有被正确预测，则将计数器的值向相反方向移动一位。

6.要让取指级和访存级的load指令不需要多等一拍，可以对流水线设计方案进行修改，引入流水线握手机制。

1、在取指级（IF）阶段：

在译码级（ID）阶段，判断当前指令是否为load指令。

如果是load指令，则将读数据的地址请求早一拍发送给数据存储器（DM）。

在译码级（ID）阶段，将读数据的地址请求发送给指令存储器（IM）。

2、在访存级（MEM）阶段：

在取指级（IF）阶段，将读数据的地址请求发送给数据存储器（DM）。

在访存级（MEM）阶段，从数据存储器（DM）中读取数据，并将数据传递给执行级（EX）阶段。