Lab4 32 位 MIPS 流水线处理器设计

1 实验目的

熟悉 MIPS 流水线 CPU 的工作原理

2 实验过程

2.1 学习 MIP 流水线处理器原理

通过将单周期处理器分解成 5 个流水线阶段来构成流水线处理器。

流水线处理器在单周期处理器基础上的改变：

(1) 指令的拆分

单周期时一条指令直接完整执行，而流水线将单条指令拆分成 5 个部分，即取指令（F）、译码（D）、执行（E）、存储（M）和写回（W）。

这正是流水线的实质，实现在每个阶段流水线中最多同时执行 5 条指令的片段。

(2) 确保控制信号与指令保持同步

因为在同一流水段中会执行不同指令的片段，为保证每个指令片段的控制信号与指令一致，在流水线中与特定指令相关的所有信号都必须通过流水线一起向前传播。

写回阶段的寄存器文件写入逻辑在执行阶段产生,故需要对 WriteReg 信号做出修正，其经过存储器和写回阶段沿流水线传递，最终和 ResultW 一起传送到寄存器文件。

(3) 如何解决数据冒险

在单周期中每条指令执行完毕后再执行其他指令，因此后一条指令所需要的地址或数据已经准备好；然而流水线执行时，当一条指令试图读取前一条指令还未写回的寄存器时会发生冲突，需要在数据通路中进行冲突解决。

RAW 类型冲突发生的具体场景为：当前一条指令需要写寄存器且后续两条指令的任何一条读这个寄存器。

同样是写寄存器，根据指令类型的不同，运用重定向或阻塞的方法来解决冲突。

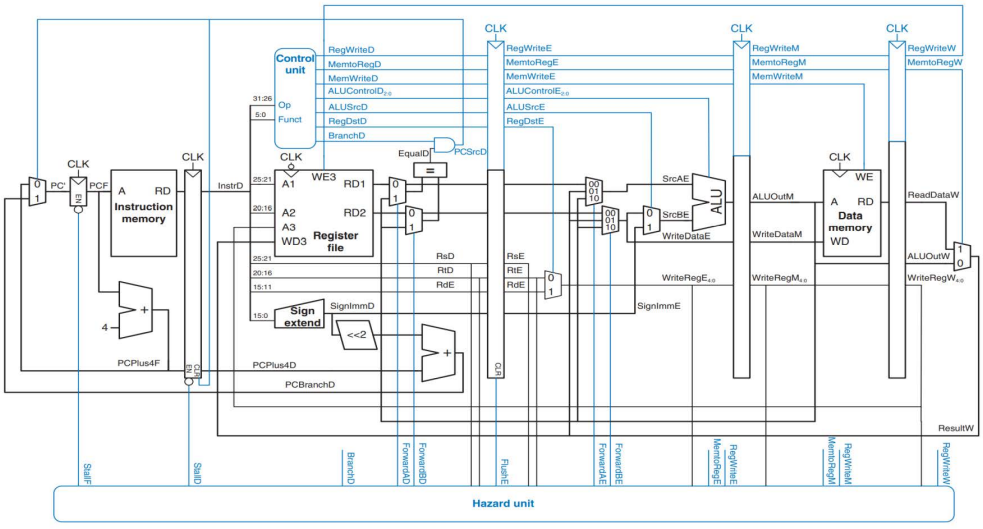
（1）重定向：当指令执行到 E 步骤时，发现所需数据与 M 或者 W 阶段的 writeregM 或者 writeregW 匹配时，可以直接使用正在传输的信号。

（2）阻塞：重定向不是任何情况都适用的，如果 E 阶段需要使用 M 阶段的输出结果（如 lw 指令）则无法重定向。此时需要将发生冲突的指令及其后指令集体“后移”，通过重复当前阶段填充一个时间周期。

(4) 如何解决控制冒险

使用提前预测分支的方法，将分支前置；分支前置又会导致数据冲突，因此在解决冲突模块中也要考虑使用重定向或阻塞来解决分支前置导致的冲突。

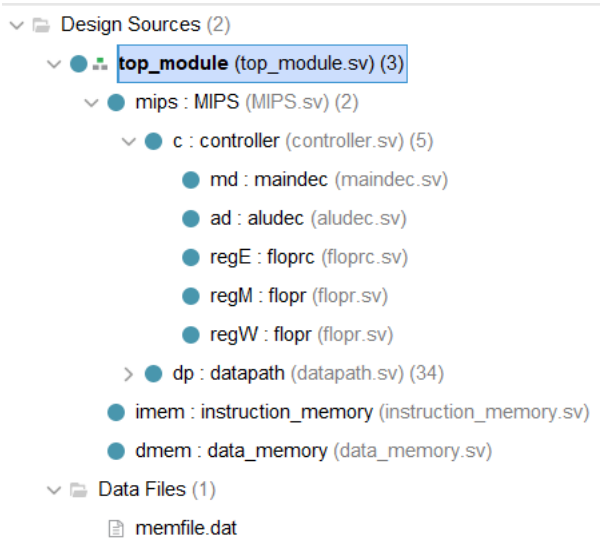
以下是能处理所有冲突的流水线处理器的原理图：



2.2 完成 MIPS 流水线处理器设计

2.2.1 添加代码框架

根据上述原理，写出代码的整体结构，如下图所示：



2.2.2 基础设计

支持 R 类型算术/逻辑指令 add、sub、and、or、slt，存储器指令 lw、sw，分支指令 beq

数据存储器模块 dmem：与单周期相同

指令存储器模块 imem：与单周期相同

MIPS 模块 MIPS：数据通路模块 datapath 控制器 controller：在主译码器模块 maindec 和 ALU 译码器模块 aludec 基础上增加流水线寄存器模块 flopr 和 floprc，保存在流水线中与特定指令相关的所有信号，确保其通过流水线一起向前传播。数据通路模块 datapath：划分为五个阶段，分别实例化对应模块，其中计算下一个 PC 在 F 阶段和 D 阶段实现，寄存器文件需要在 D 阶段和 W 阶段用到。

2.2.3 扩展 addi

增加的操作：

增加零扩展元件 zeroext，接受 16 位输入，并通过添加 16 个零将它们扩展到 32 位；

增加扩展信号 SorZD，当指令操作码 op 为 andi 类型时，需要进行零扩展，设置扩展 SorZD 为 1，设置 ALU 操作码 aluop 信

号为 2'b11，通过 ALU 译码器设置 ALU 控制信号 alucontrol 为 3'b000，在 E 阶段进行与操作；

将零扩展元件 zeroext 的输出零扩展数据 zeroImmD 和符号扩展器 signext 的输出符号扩展数据 signImmD 合并到一个 2 输入多路复用器 mux2 中，为多路复用器增加扩展信号 SorZD，当这个信号为 1 时则立即数为零扩展，当其为 0 时则立即数为符号扩展；

将该 2 输入多路复用 mux2 的输出扩展数据 extImmD 作为替代原来的符号扩展数据 signImmD，传递给 E 阶段的 signImmE，作为 ALU 源操作数 SrcBE 可选值之一。

2.2.4 处理冒险：hazard 模块

以下是对每种可能的冒险情况的处理方法：

数据冒险：

E 阶段指令中一个源寄存器与 M/W 阶段的目标寄存器相匹配——重定向

lw 指令目标寄存器与 D 阶段源操作数相匹配——重定向

分支冒险：

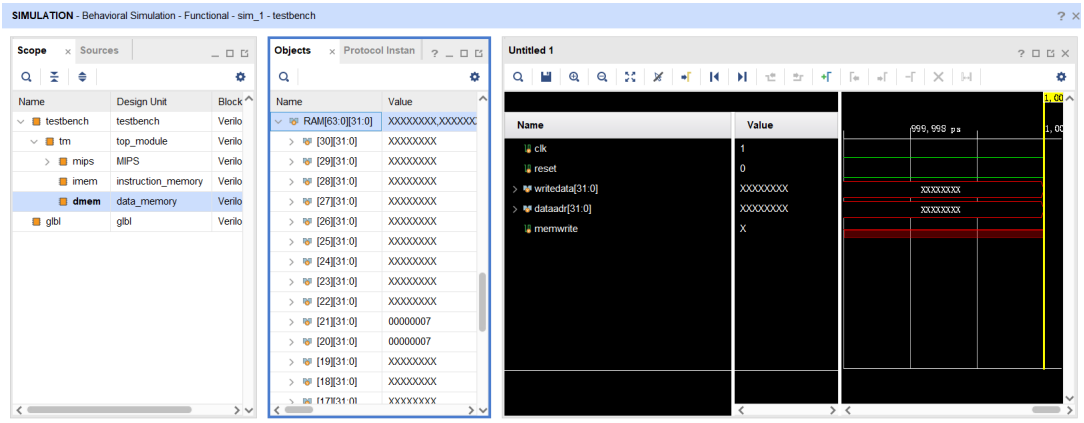
beq 指令源寄存器依赖 M 阶段指令 ALU 的结果——重定向

beq 指令源寄存器依赖 E 阶段指令 ALU 的结果——阻塞

beq 指令源寄存器依赖 M 阶段 lw 指令的结果——阻塞

2.3 仿真测试

用与之间实验相同的 memfile.dat 进行仿真测试，结果如下图：



可以看到在内存中的 21 号地址（因为寻址方式与书中给出方式不同，所以地址会除以 4）处存入了 7，表明操作正确。

3 实现体会

在本次实验中，我们成功地实现了一个 32 位的 MIPS 流水线处理器。这个过程需要精准的步骤和精妙的设计。

1. 拆分指令与同步控制信号：我们将单周期处理器拆分成了五个阶段：取指（F）、译码（D）、执行（E）、存储（M）和写回（W）。通过这种方式，我们显著提高了处理效率。当然，为了确保每条指令在流水线中的控制信号保持同步，我们必须将这些控制信号与指令一同向前传播。

2. 解决数据冒险：数据冒险是流水线处理器中的一个关键问题。在单周期处理中，每条指令执行完毕后再执行下一条指令，因此不会有数据冲突。但是在流水线中，当一条指令试图读取前一条指令还未写回的寄存器时，就会发生冲突。我们采用了重定向和阻塞的方法来解决这个问题。

3. 解决控制冒险：控制冒险主要出现在分支指令中。为了减少控制冒险的影响，我们使用了提前预测分支的方法，将分支前置。这虽然解决了部分问题，但也带来了新的数据冲突。我们需要在冲突解决模块中进一步使用重定向或阻塞的方法来解决这些问题。

通过这次实验，我不仅在这个过程中深刻体会到了流水线处理器的设计规范，也锻炼了自己解决复杂问题的能力。