实验六：流水线CPU设计

实验内容

使用Verilog设计实现流水线CPU。

文件内容

- ./实验报告.pdf：该实验的实验报告。
- ./code文件夹：储存实验相关代码的文件夹。
    - ./code/Pipeline.sv：流水线CPU。
    - ./code/alu.v：算术/逻辑运算模块。
    - ./code/ControllerUnit.v：控制器模块。
    - ./code/DataMemory.v：内存模块。
    - ./code/ProgramCounter.v：指令寄存器模块。
    - ./code/GeneralPurposeRegisters.v：寄存器模块。

流水线竞争

根据实验要求，报告中仅讲解对于流水线竞争的处理。

（**1**）结构冒险

- 从内存中取指和取数的冒险

  将数据和指令存放到两个不同的存储器中，将二者分开，消除冒险。

- 读寄存器和写寄存器的冒险

  在时钟的下降沿进行寄存器的写入操作，即在时钟周期的中间进行修改。由于读寄存器是组合逻辑，对寄存器进行修改后，仍然有半个时钟周期的时间用于读取新的值，保证读到的寄存器的数值是正确的。

（**2**）数据冒险

每级流水线寄存器设置一个信号$[1:0]\ ready$：

- $ready=0$表示这级流水线需要计算rd寄存器的值，但是还没计算出来。

- $ready = 1$表示这级流水线需要计算rd寄存器的值，已经计算出来。
- $ready = 2$表示这级流水线不需要计算rd寄存器的值，即这级流水线寄存器中的rd信号无意义。

对于ADDU、SUBU、ORI等ALU操作相关的指令，在经过EX段之后，已经能计算得到rd寄存器的值，这时候把$ready$设置为1。对于LW指令，需要在MEM段访问内存之后才能得到寄存器的值，所以需要经过MEM段后才能把$ready$设置为1。

在WB段中，只有当$ready = 1$时才将结果写入rd中。

在EX阶段，需要使用寄存器的值时，

1. 到EX/MEM中寻找。如果$ready$为2，则跳过这级流水线寄存器。否则，如果rd为所需要的寄存器，且$ready$为1，那么直接获得这个结果；如果rd为所需要的寄存器，且$ready$为0，说明该寄存器最新的值还没有计算出来，在EX段加一个气泡进行等待。
2. 到MEM/WB中寻找。如果$ready$为2，则跳过这级流水线寄存器。否则，如果rd为所需要的寄存器，且$ready$为1，那么直接获得这个结果；如果rd为所需要的寄存器，且$ready$为0，说明该寄存器最新的值还没有计算出来，在EX段加一个气泡进行等待。
3. 如果上述两级流水线寄存器都没有找到，则使用从ID段读取得到的值。

   特别的，如果增加了气泡，需要重新从寄存器中读取数据，防止有的寄存器的值已经经过WB段写回寄存器后，不再出现在流水线寄存器上，以至于使用了错误的值进行计算。

在ID阶段，需要使用寄存器的值时（处理控制指令），

1. 到ID/EX中寻找。如果$ready$为2，则跳过这级流水线寄存器。否则，如果rd为所需要的寄存器，且$ready$为1，那么直接获得这个结果；如果rd为所需要的寄存器，且$ready$为0，说明该寄存器最新的值还没有计算出来，在ID段加一个气泡进行等待。

   这里从ID/EX段寻找值，是为了防止跳转指令需要用到的前一条指令的计算结果，需要检查对应寄存器的最新的值是否已经被计算出来
2. 到EX/MEM中寻找。如果$ready$为2，则跳过这级流水线寄存器。否则，如果rd为所需要的寄存器，且$ready$为1，那么直接获得这个结果；如果rd为所需要的寄存器，且$ready$为0，说明该寄存器最新的值还没有计算出来，在ID段加一个气泡进行等待。
3. 到MEM/WB中寻找。如果$ready$为2，则跳过这级流水线寄存器。否则，如果rd为所需要的寄存器，且$ready$为1，那么直接获得这个结果；如果rd为所需要的寄存器，且$ready$为0，说明该寄存器最新的值还没有计算出来，在ID段加一个气泡进行等待。

4. 如果上述两级流水线寄存器都没有找到，则使用从ID段读取得到的值。

   特别的，如果增加了气泡，需要重新从寄存器中读取数据，防止有的寄存器的值已经经过WB段写回寄存器后，不再出现在流水线寄存器上，以至于使用了错误的值进行计算。

特别的，如果需要找的是0号寄存器，由于计算过程中可能会计算出其他的值，但是不会真正写入寄存器，所以不从旁路中寻找。

（**3**）控制冒险

由于存在分支延时槽，所以读入控制指令后，下一条也是需要执行的指令，可以正常进入IF段。

此时，控制指令已经到达ID段，可以从寄存器（或者旁路）获取对应的数据，通过判断可以确定再下一条指令的地址，这样能够保证每次取到的指令都是需要执行的指令。