单周期 CPU 设计

目标

设计并实现单周期 MIPS CPU，支持以下 MIPS 指令：

- addu：寄存器加法
- subu：寄存器减法
- ori：立即数或
- lw：对齐加载
- sw：对齐存储
- beq：相等时跳转
- lui：加载立即数到高位
- jal：函数调用
- jr：寄存器跳转
- 附加：syscall：执行到 syscall 指令时使用 $finish 终止仿真

在单周期 CPU 中，我们不处理分支延迟槽（Branch Delay Slot），即与 x86 类似，条件跳转指令会在执行后立即生效。参考 Mars 的配置与之前相同。

具体的指令编码与行为请详见「**MIPS32 指令集.pdf**」，或自行寻找其他资料。

说明

推荐将每个独立的模块写在单独的文件中，并在顶层实例化各个模块：

- **ProgramCounter**：维护 PC 值
  - 功能与上次作业中一致
  - PC 值初始化为 0x00003000，被自动截断后指向 InstructionMemory 的 0 位置
- **ControllerUnit**：对指令进行解码并输出控制信号
  - 输出控制信号的逻辑较为复杂，建议使用 always 组合逻辑块实现硬布线
  - 建议学习 SystemVerilog，可以将控制信号组合在结构体中
- **GeneralPurposeRegisters**：维护通用寄存器值
  - 寄存器与数据存储器相同，使用 reg 数组实现，在 reset 信号时清零
  - 输入：要读取的寄存器编号 1、编号 2 以及要写入的寄存器编号
  - 输入：是否开启写入，要写入的值
  - 输出：读取到的值
  - 初始化为全零
- **ArithmeticLogicUnit**：进行算术/逻辑运算
  - 与之前作业中类似，建议在控制信号中自行定义各种运算的操作码
- **DataMemory**：读写数据存储
  - 大小为 4 KiB，即 1024 字
  - 初始化为全零
- **InstructionMemory**：读指令存储
  - 大小为 4 KiB，即 1024 字
  - 指令存储为只读，在仿真启动时初始化，见下文讲解

请在顶层根据控制信号将数据通路连接，并处理跳转指令。
顶层模块接受 clock 和 reset 两个输入信号。

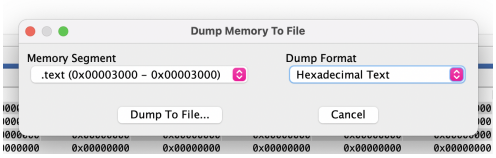
测试

在测试时，请使用 Mars 汇编器将 MIPS 汇编代码转化为机器码，提供给 InstructionMemory 作为输入。

使用以下命令调用 Mars 的汇编功能，对 mips.asm 中的代码进行汇编，结果输出十六进制字符串格式的机器码到 code.txt 文件。
注：Windows 下可能需要指定 java 程序的绝对路径。

```
java -jar Mars.jar dump .text HexText code.txt a nc mc CompactDataAtZero mips.asm
```

或者在 Mars 图形界面中打开 MIPS 汇编代码文件，点击编译后选择 File – Dump memory，并选择十六进制文本格式导出：



之后将 mips.asm 放置到某个路径下，即可在 InstructionMemory 中读入它（注意：请使用绝对路径）：

```
initial
        // Linux / macOS:
        $readmemh("/tmp/code.txt", memory);
        // Windows:
        // $readmemh("D:\\code.txt", memory);
```

运行完成（可以使用 syscall 指令来终止运行）后将你的 CPU 运行结果与 Mars 的运行结果相比较（主要是比较 DataMemory 中的内容），如果不一致则说明有问题。可以通过逐周期运行（即逐指令运行）的方式来调试。

TestBench 文件附带在 mips_test.v 中，请注意修改其中顶层模块的模块名。

注：单周期 CPU 的正确性以运行后 DataMemory 的内容是否正确为准。Vivado 中可以直接在运行后展开 DataMemory 模块进行查看。如果其他仿真环境中不方便直接查看，可在 $finish 之后在顶层使用一个 for 循环将 DataMemory 中数据（Verilog 在仿真时支持 dm.data[i] 跨元件访问）用 $display 输出，以检查运行结果。

提交

请提交所有自己编写的代码文件。实验报告中只需要描述自己对控制信号的设计即可。

助教将会在实验课上现场检查 mips1.asm 汇编程序的运行结果。