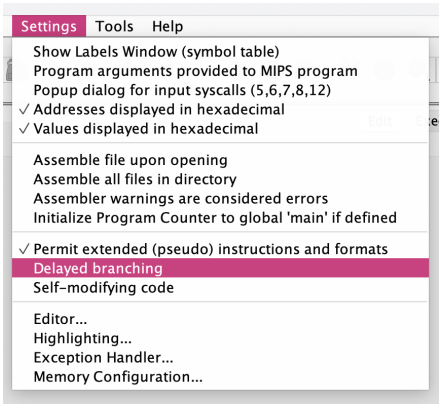
流水线 CPU 设计

目标

设计并实现五级流水线 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
- j：跳转
- 附加：syscall：执行到 syscall 指令时使用 $finish 终止仿真

在流水线 CPU 中，我们需要处理分支延迟槽（Branch Delay Slot）。请在 Mars 中打开该选项：



说明

你需要实现五级流水线（IF-ID-EX-MEM-WB），并实现完整的旁路转发。执行同一程序时，你的实现所用时钟周期应当仅与参考时限相差常数级别。

每一级流水线的结果，应当在每个时钟周期结束时，储存在这一级的**流水线寄存器**中。也就是说，每一时刻，流水线寄存器储存了该级流水线**上个周期**的结果。除该级流水线上个周期的结果外，为了方便调试和检查，应当额外存储其**上个周期所运行的指令**以及**所处的 PC 值**。

在使用旁路转发处理冲突时，应当转发来自某一级流水线寄存器的值，而非具体元件的值。

建议实现一种通用的模式来处理「旁路转发」与「阻塞」，而不要尝试枚举具体的每一种转发与阻塞的情况，后者将大大增加代码复杂度。

建议将控制信号封装为 struct，并沿着流水线传递，并将每一级流水线寄存器封装为 struct。

请牢记阻塞赋值（=）与非阻塞赋值（<=）的特性。

测试

为了确保能够检查出所有错误，并方便地定位出错位置，我们记录每次寄存器写入与内存写入。

在写入寄存器时添加如下 $display 语句（请将该指令所在的 PC 传入通用寄存器组件）：

```
$display("@%h: $%d ⩽ %h", programCounter, registerId, dataWrite);
```

在写入内存时添加如下 $display 语句（其中 address 为完整的 32 位地址；请将该指令所在的 PC 传入数据存储器组件）：

```
$display("@%h: *%h ⩽ %h", programCounter, address, dataWrite);
```

执行时，Vivado 控制台或其它仿真工具的输出中应当含有如下内容：

```
@00003000: $31 ⩽ 00003000
@00003004: $ 2 ⩽ 00000000
@00003008: $ 2 ⩽ 00000000
@0000300c: $ 0 ⩽ 00030000
@00003010: $ 0 ⩽ 00000000
@00003014: $ 1 ⩽ 00040000
@00003018: *00000004 ⩽ 00003000
@0000301c: $ 1 ⩽ 00006000
@00003020: *00000008 ⩽ 00003000
@00003024: $ 1 ⩽ 00003000
```

作为参考，请下载带有输出信息的 Mars（请不要使用该版本 Mars 的图形界面），并使用命令行执行你的汇编程序：

```
java -jar Mars.jar db nc 500 ae2 mc CompactDataAtZero <你的汇编程序>.asm
```

对比你的输出信息与 Mars 的输出信息，第一次出现的不一致的行即为你的出错位置。

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流水线竞争的处理即可。

测试程序将在之后下发。