# 计算机组成原理-P6 Verilog 流水线 Plus 实验报告

## 一、模块规格

### 1. IFU

取指令模块，位于 IF 级，在流水线 CPU 中内部仅包含寄存器 PC 以及指令存储器 IM，接受来自 nPC 的输入并更新 PC

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| clk | I | 时钟信号 |
| reset | I | 重置信号 |
| IFU_i_En | I | PC 寄存器使能，用来控制指令冲突时暂停 |
| IFU_i_nPC[31:0] | I | 跳转目标地址，针对 JR 等指令 |
| IFU_o_Instr[31:0] | O | 当前 PC 所指指令 |
| IFU_o_PC [31:0] | O | 当前 PC 的值 |

功能定义表

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 重置 | reset 置 1 时，PC 寄存器复位至 0x00003000 |
| 2 | 输出指令 | Instr 为 PC 所指处指令，PC <= nPC |

### 2. CMP

比较器模块，接受 rs、rt，根据 Mode 信号进行比较并输出结果。

| 信号名 | 方向 | 描述 |
|---|---|---|
| CMP_i_RsIn[31:0] | I | Rs 输入 |
| CMP_i_RtIn[31:0] | I | Rt 输入 |
| CMP_i_Mode[3:0] | I | CMP 比较模式：1-相等，2-不等，3-小于等于 0，4-大于等于 0，5-小于 0，6-大于 0 |
| CMP_o_Output | O | 比较结果 |

### 3. NPC

PC 值更新模块，位于 D 级，接受当前 PC 输入（直接从 IF 级连入）、控制信号输出和 Condition（条件跳转的条件），输出新的 PC 值。

| 信号名 | 方向 | 描述 |
|---|---|---|
| NPC_i_PC[31:0] | I | 当前 PC 值输入 |
| NPC_i_Immediate[25:0] | I | 输入的立即数（包括 J 系列的 26 位和 B 系列的 16 位） |
| NPC_i_RegAddr[31:0] | I | JR 系列指令输入的地址 |
| NPC_i_Mode[2:0] | I | 指示 nPC 如何计算，详见 Controller |
| NPC_i_Condition | I | 条件跳转时条件的输入 |
| NPC_o_nPC[31:0] | O | 输出 nPC 值 |

功能：确定 PC 寄存器的下一个值

## 4. GRF

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| clk | I | 时钟信号 |
| reset | I | 重置信号 |
| GRF_i_WEnable | I | 写使能信号 |
| GRF_i_RAddr1[4:0] | I | 读功能所读寄存器地址 1 |
| GRF_i_RAddr2[4:0] | I | 读功能所读寄存器地址 2 |
| GRF_i_WAddr[4:0] | I | 写功能所写寄存器地址 |
| GRF_i_WData[31:0] | I | 写功能写入数据 |
| GRF_i_PC[31:0] | I | $display 使用的 PC 数据 |
| GRF_o_RData1[31:0] | O | 读寄存器数据 1 |
| GRF_o_RData2[31:0] | O | 读寄存器数据 2 |

功能定义表

| 序号 | 功能名称 | 描述 |
|---|---|---|
| 1 | 重置 | reset == 1 时重置 GRF，所有寄存器归零 |
| 2 | 读取寄存器数据 | clk 接时钟信号时，GRF_o_RDdata1/2 输出寄存器编号为 GRF_i_RAddr1/2 的寄存器的值 |
| 3 | 写入寄存器 | GRF_i_WEnable == 1 时，将 GRF_i_WData 处数据在下一个时钟上升沿存入编号为 GRF_i_WAddr 的寄存器 |

## 5. ALU

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| ALU_i_Operand1[31:0] | I | 操作数 1 |
| ALU_i_Operand2[31:0] | I | 操作数 2（具体由 ALUOp2Datapath 元件确定） |
| ALU_i_Operation[3:0] | I | 具体操作，0：加， 1：减， 2: 或，3:lui 操作，其他待定 |
| ALU_o_Result[31:0] | O | 运算结果 |
| ALU_o_isOverflow | O | ADD 和 SUB 的结果是否溢出 |

功能定义表

| 序号 | 功能名称 | 描述 |
|---|---|---|
| 1 | 运算 | 根据 ALU_Operator 指定的操作进行运算，结果输出至 Result。具体操作：0：加（无溢出检查），1：加，2：减（无溢出检查），3：减，4: 与，5：或，6：异或，7：或非，8：逻辑左移，9：逻辑右移，10：算术右移，11：小于时置位，12：小于时置位（无符号），13：lui 操作，其他待定 |
| 2 | 判断结果是否溢出 | 运算结果为 0 时，ALU_Overflow 置 1 |

6. **MDU**

乘除法单元，端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| Clk | I | 时钟信号 |
| Reset | I | 重置数据，清零 Hi/Lo |
| MDU_i_Operand1[31:0] | I | 操作数 1 |
| MDU_i_Operand2[31:0] | I | 操作数 2 |
| MDU_i_Operation[4:0] | I | 具体操作，1-乘法，2-无符号乘法，3-除法，4-无符号除法，5-mthi，6-mtlo，7-read |
| MDU_o_Busy | O | 是否处于 Busy 延迟状态 |
| MDU_o_Hi[31:0] | O | Hi 寄存器 |
| MDU_o_Lo[31:0] | O | Lo 寄存器 |

7. **DM**

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| clk | I | 时钟信号 |
| reset | I | 重置数据，清零 RAM |
| DM_i_Addr[31:0] | I | 操作地址 |
| DM_i_WData[31:0] | I | 待写入数据 |
| DM_i_WEnable | I | 是否写入数据 |
| DM_i_WMode[1:0] | I | 操作位宽，0 -- word, 1 -- half, 3 -- byte, 4 -- byte_unsigned, 5 -- half_unsigned |
| DM_i_PC | I | $display 用 PC 数据 |
| DM_o_Rdata[31:0] | O | 操作地址处的数据 |

功能定义表

| 序号 | 功能名称 | 描述 |
|---|---|---|
| 1 | 读取 | 以指定位宽读取指定位置数据 |
| 2 | 写入 | 以指定位宽在指定位置写入数据 |

注：对于不是以字为宽度的 IO 操作，输出采用符号延拓

8. **EXT**

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| EXT_i_Input[15:0] | I | 输入数据 |
| EXT_i_Mode[1:0] | I | 操作模式： 0 - 无符号延拓 32， 1 - 有符号延拓 32 |
| EXT_o_Output[31:0] | I | 输出数据 |

功能定义表

| 序号 | 功能名称 | 描述 |
|---|---|---|
| 1 | 无符号延拓 32 位 | EXT_Mode == 0 时，将 EXT_Input 做 32 位无符号延拓 |
| 2 | 有符号延拓 32 位 | EXT_Mode == 1 时，将 EXT_Input 做 32 位有符号延拓 |

# 二、控制器设计

## 1. 概述

该部分介绍 Controller 及各 MUX 的详细信息。控制器设计仍然分为两部分：对指令的识别以及指令被识别后控制信号的生成。两过程由一个 31 位的 instr_type 变量连接，每种指令（功能完全一致，数据可以不同）对应一个唯一的值。内体总体端口定义如下，具体信号功能请参见下文描述：

| 信号名 | 方向 | 描述 |
|---|---|---|
| Instr[31:0] | I | 指令 |
| instr_type[31:0] | O | 当前指令为何种指令（用于冒险控制） |
| NPC_Mode[2:0] | O | nPC 计算方法 |
| CMP_Mode[3:0] | O | CMP 比较方法 |
| GRF_WEnable | O | 寄存器写使能信号 |
| ALU_Operation[7:0] | O | ALU 操作信号 |
| MDU_Operation[4:0] | O | MDU 操作信号 |
| DM_WEnable | O | 内存写使能 |
| DM_Mode[1:0] | O | 操作位宽 |
| EXT_Mode[1:0] | O | Extender 模式 |
| MUX_RegWAddr_Sel[2:0] | O | 寄存器写地址选择 |
| MUX_RegWData_Sel[2:0] | O | 寄存器写数据选择 |
| MUX_ALUOp1_Sel[2:0] | O | ALU 操作数 1 选择信号 |
| MUX_ALUOp2_Sel[2:0] | O | ALU 操作数 2 选择信号 |
| MUX_EALU_OData_Sel[2:0] | O | E 级 ALU 写入流水线寄存器的信号选择 |

## 2. 相关各 MUX 控制信号说明

a) MUX_RegWAddr: 根据 MUX_RegWAddr_sel 的选择输出对应数据

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| MUX_RegWAddr_i_Sel[2:0] | I | 选择信号：0 --- rd， 1 --- rt，2 --- $ra，其他待定 |
| MUX_RegWAddr _i_rdIn[4:0] | I | rd 输入 |
| MUX_RegWAddr _i_rtIn[4:0] | I | rt 输入 |
| MUX_RegWAddr _o_Output[4:0] | O | 输出 |

b) RegWData：根据 RegWData_Sel 的选择输出对应数据

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| MUX_RegWData_i_Sel[2:0] | I | 选择信号：0 - alu 结果， 1 - mem 数据，2 - PC+4，其他待定 |
| MUX_RegWData_i_ALUIn[31:0] | I | ALU 结果 |
| MUX_RegWData_i_memIn[31:0] | I | MEM 结果 |
| MUX_RegWData_i_linkIn[31:0] | I | PC+4 输入 |
| MUX_RegWData_o_Output[31:0] | O | 输出 |

c) MUX_ALUOp2: 根据 MUX_ALUOp2Data_Sel 的选择输出对应数据

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| MUX_ALUOp2_i_Sel[2:0] | I | 选择信号，0 --- RegData2，1 --- EXTender，2,3 --- 待定 |
| MUX_ALUOp2_i_regIn[31:0] | I | reg 数据输入 |
| MUX_ALUOp2_i_EXTIn[31:0] | I | ext 数据输入 |
| MUX_ALUOp2_o_Output[31:0] | O | 输出 |

d) MUX_ALUOp1: 根据 MUX_ALUOp1Data_Sel 的选择输出对应数据

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| MUX_ALUOp1_i_Sel[2:0] | I | 选择信号，0 - rs，1 - Shamt |
| MUX_ALUOp1_i_RsIn[31:0] | I | Rs 数据输入 |
| MUX_ALUOp1_i_ShamtIn[31:0] | I | Shamt 数据输入 |
| MUX_ALUOp1_o_Output[31:0] | O | 输出 |

e) MUX_EALU_OData: 根据 MUX_EALU_OData_Sel 的选择输出对应数据

端口定义表

| 信号名 | 方向 | 描述 |
|---|---|---|
| MUX_EALU_OData _i_Sel[2:0] | I | 选择信号，0 – ALU, 1 – Hi，2 – Lo |
| MUX_EALU_OData_i_ALUIn[31:0] | I | ALU 数据输入 |
| MUX_EALU_OData_i_HiIn[31:0] | I | Hi 数据输入 |
| MUX_EALU_OData_i_LoIn[31:0] | I | Lo 数据输入 |
| MUX_ALUOp2_o_Output[31:0] | O | 输出 |

3. **指令识别**

指令识别由一个 always 块描述的组合逻辑实现。在 case(Instr[31:26])和 case(Instr[5:0])内，为 instr_type 赋对应当前指令的值，真值表如下：

功能：译码 --- 当检测到某指令时 instr_type 输出其对应信号

（见压缩包中所附 Excel 表格）

4. **ControlSignal**

端口定义表及信号介绍

| 信号名 | 方向 | 描述 |
|---|---|---|
| instr_type | O | 当前指令类型，见指令识别部分 |
| NPC_Mode | O | nPC 控制, 0 – 顺序，1 – 条件分支，2 – 跳转， 3 – 跳至寄存器 |
| CMP_Mode[3:0] | O | CMP 比较模式：1-相等，2-不等，3-小于等于 0，4-大于等于 0，5-小于 0，6-大于 0 |
| GRF_WEnable | O | 寄存器写使能信号，0：GRF 不写入，1：GRF 写入 |
| ALU_Operation[7:0] | O | ALU 操作信号，0：加（无溢出检查），1：加，2：减（无溢出检查），3：减，4: 与，5：或，6：异或，7：或非，8：逻辑左移，9：逻辑右移，10：算术右移，11：小于时置位，12：小于时置位（无符号），13：lui 操作，其他待定 |
| MDU_Operation[3:0] | O | MDU 操作信号，1-乘法，2-无符号乘法，3-除法，4-无符号除法，5-mthi，6-mtlo，7-read |
| DM_WEnable | O | 内存写使能，0：DM 不写入，1：DM 写入 |
| DM_Mode[1:0] | O | 位宽，0：字，1：半字，2：字节，3：无符号字节，4：无符号半字 |
| EXT_Mode | O | Extender 模式，0：无符号拓展至 32 位，1：有符号拓展至 32 位 |
| MUX_RegWAddr_Sel[2:0] | O | 寄存器写地址选择，0：rd，1：rt，2：$ra |
| MUX_RegWData_Sel[2:0] | O | 寄存器写数据选择，0：ALU，1：Mem，2：PC+4 |
| MUX_ALUOp1_Sel[2:0] | O | ALU 操作数 1 选择信号，0：rs，1：Shamt |
| MUX_ALUOp2_Sel[2:0] | O | ALU 操作数 2 选择信号，0：rt，1：EXT |
| MUX_EALU_OData_i_Sel[2:0] | I | E 级 ALU 输出写入寄存器选择信号，0 – ALU, 1 – Hi，2 – Lo |

功能：根据输入指令信号输出对应控制信号

（见压缩包中所附 Excel 表格）

# 四、流水线结构及转发、阻塞

## 1. 流水线总体结构

流水线分为 5 个阶段，分别为 IF、ID、EX、MEM、WB 阶段，分别执行取指令、取 GRF 及立即数拓展、ALU 运算、写内存和数据写回寄存器的功能。流水线各阶段之间由流水线寄存器（PReg）连接，具体数据域如下：

| 数据域 | 说明 |
|---|---|
| Instr[31:0] | 指令 |
| PC[31:0] | PC 值 |
| rsData[31:0] | rs 域指明的寄存器所存放的数据 |
| rtData[31:0] | rt 域指明的寄存器所存放的数据 |
| extData[31:0] | EXT 输出的数据 |
| ALUResult[31:0] | ALU 运算结果 |
| memData[31:0] | 读内存读出的数据 |
| RegWData[31:0] | 若指令回写寄存器，则回写值产生时将被存放至此处 |

除此之外，还有数个控制信号，如下：

| 控制信号 | 说明 |
|---|---|
| clk | 时钟信号 |
| reset/clear | 清空 PReg |
| PReg_i_Enable | PReg 寄存器写使能，为 1 时才更新 |

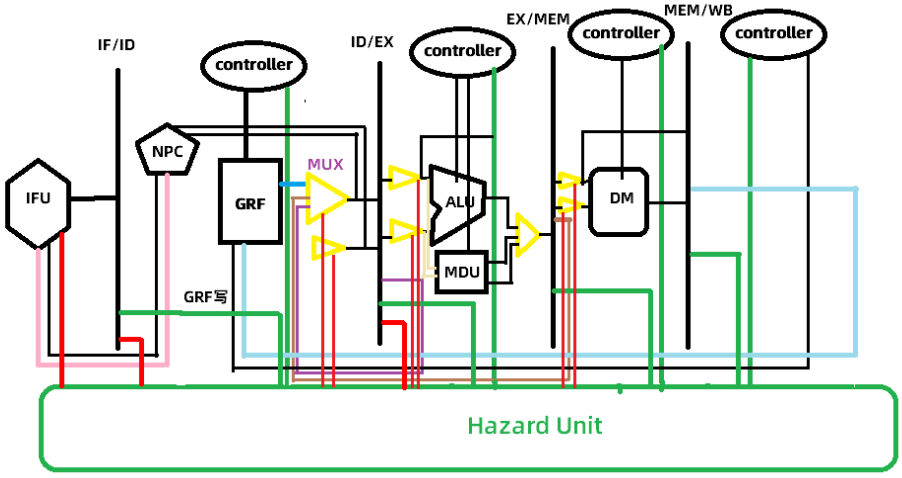
除了某些极特殊特殊情况（为了方便，ID 级 NPC 部件接的是 IF 级 PC）以及转发时的数据，各流水线阶段只需要读取 PReg 内容。

每个流水线阶段都有一个 Controller，4 个 Controller 完全一致，由同一个元件例化而来，只连接本阶段所需要的线。

流水线的冒险由一个专门的冒险单元（Hazard Unit，HU）控制，这个单元接受完成冒险控制功能所需的流水线各阶段输入，输出转发相关 MUX（FMUX）的控制信号以及与阻塞有关的控制信号。具体接口如下：

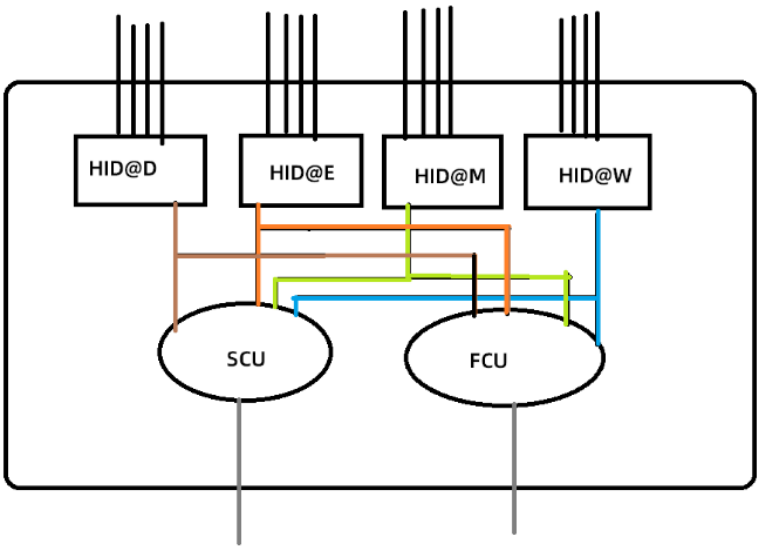
| 信号名 | 方向 | 功能 |
| --- | --- | --- |
| HU_i_D_Instr[31:0] | I | D 级指令 |
| HU_i_D_Instr_Type[31:0] | I | D 级指令类型 |
| HU_i_D_RegWAddrSel[4:0] | I | D 级写寄存器地址 |
| HU_i_D_GRFWEnable | I | D 级指令寄存器写使能 |
| HU_i_E_Instr[31:0] | I | E 级指令 |
| HU_i_E_Instr_Type[31:0] | I | E 级指令类型 |
| HU_i_E_RegWAddrSel[4:0] | I | E 级写寄存器地址 |
| HU_i_E_GRFWEnable | I | E 级指令寄存器写使能 |
| HU_i_E_MDUBusy | I | E 级 MDU 的 Busy 信号 |
| HU_i_M_Instr[31:0] | I | M 级指令 |
| HU_i_M_Instr_Type[31:0] | I | M 级指令类型 |
| HU_i_M_RegWAddrSel[4:0] | I | M 级写寄存器地址 |
| HU_i_M_GRFWEnable | I | M 级指令寄存器写使能 |
| HU_i_W_Instr[31:0] | I | W 级指令 |
| HU_i_W_Instr_Type[31:0] | I | W 级指令类型 |
| HU_i_W_RegWAddrSel[4:0] | I | W 级写寄存器地址 |
| HU_i_W_GRFWEnable | I | W 级指令寄存器写使能 |
| HU_o_F_IFUEnable | O | IFU 使能，用于阻塞 |
| HU_o_D_PRegEnable | O | IF/ID 级 PReg 使能信号，用于阻塞 |
| HU_o_E_PRegClear | O | ID/EX 级 PReg 复位信号，用于阻塞 |
| HU_o_FwdD_Rs[1:0] | O | ID 级 rs 转发控制信号 |
| HU_o_FwdD_Rt[1:0] | O | ID 级 rt 转发控制信号 |
| HU_o_FwdE_Rs[1:0] | O | EX 级 rs 转发控制信号 |
| HU_o_FwdE_Rt[1:0] | O | EX 级 rt 转发控制信号 |
| HU_o_FwdM_Rs[1:0] | O | MEM 级 rs 转发控制信号 |
| HU_o_FwdM_Rt[1:0] | O | MEM 级 rt 转发控制信号 |

以下为总体结构示意图：




## 2. 流水线冒险单元总体结构

流水线冒险单元接受各级输入后，将每一级的输入分别连接至一个 HID（Hazard Instruction Decoder）的实例，由 HID 负责解析有关信息。HID 的输出经过处理（把 Tnew@D 处理成真正的 Tnew），接入 SCU（Stall Control Unit）和 FCU（Forward Control Unit），输出各转发控制信号以及暂停信号。总体结构如下图：



### 3. 冒险指令解码单元

HID 的输出用于对冒险的处理，具体接口如下：

| 信号 | 方向 | 说明 |
|---|---|---|
| HID_i_Instr | I | 指令 |
| HID_i_Instr_Type | I | 指令类型 |
| HID_i_RegWAddrSel | I | 写寄存器地址选择 |
| HID_o_Rs | O | rs |
| HID_o_Rt | O | rt |
| HID_o_TuseRs | O | rs 的 Tuse |
| HID_o_TuseRt | O | rt 的 Tuse |
| HID_o_TnewD | O | 指令处在 D 级时的 Tnew |
| HID_o_RegWAddr | O | 指令所写寄存器的地址（若有） |
| HID_o_MDU_Usage | O | 指令对 MDU 的使用情况 |

### 3. 暂停控制单元

| 信号 | 方向 | 说明 |
|---|---|---|
| SCU_i_D_Rs | I | D 级指令的 rs |
| SCU_i_D_Rt | I | D 级指令的 rt |
| SCU_i_D_TuseRs | I | D 级指令 rs 的 Tuse |
| SCU_i_D_TuseRt | I | D 级指令 rt 的 Tuse |
| SCU_i_D_MDU_Usage | I | D 级指令 MDU 使用情况 |
| SCU_i_E_Tnew | I | E 级指令的 Tnew |
| SCU_i_E_WAddr | I | E 级指令要写的寄存器值 |
| SCU_i_E_MDU_Usage | I | E 级指令 MDU 使用情况 |
| SCU_i_E_MDU_Busy | I | MDU 的 Busy 信号 |
| SCU_i_M_Tnew | I | M 级指令的 Tnew |
| SCU_i_M_WAddr | I | M 级指令要写的寄存器值 |
| SCU_o_Stall | I | 是否需要暂停 |

设置不写寄存器的指令 Tnew 为 0，就不会无谓的暂停。暂停条件：D 级 rs/rt 为 E/M 级要写的寄存器（W 有 GRF 内部转发），且不是 0 号寄存器，且来不及转发（Tuse < Tnew），以及新增加的乘除法单元相关条件。

### 3. 转发控制单元

功能：输出 6 个转发 MUX 所需的控制信号

输入接口：D/E/M/W 流水寄存器中指令的 rs/rt/写 Reg 地址/写使能，输出六个 FMUX 控制信号。只要流水线后一阶段指令生成了结果，且 rs/rt 与前方所写地址相同，就转发。

## 四、测试程序及思考题中测试数据相关部分

首先测试新增加指令基本正确性并尝试一些冒险，具体单条指令在此省略（基本和下文自动生成的指令类似）

其次测试乘除法指令基本正确性，确保新运算部件/乘法阻塞合理：

```
li $3, 1234567
li $4, 2469135
mult $3, $4
mfhi $5
mfhi $6
multu $3, $4
mfhi $5
mfhi $6
div $3, $4
mfhi $5
mfhi $6
divu $3, $4
mfhi $5
mfhi $6

mthi $3
mtlo $4

li $3, -1234567
li $4, 126
mult $3, $4
mfhi $5
mfhi $6
multu $3, $4
mfhi $5
mfhi $6
div $3, $4
mfhi $5
mfhi $6
divu $3, $4
mfhi $5
mfhi $6

li $3, 1234567
li $4, -126
mult $3, $4
mfhi $5
mfhi $6
```

```
multu $3, $4
mfhi $5
mfhi $6
div $3, $4
mfhi $5
mfhi $6
divu $3, $4
mfhi $5
mfhi $6

li $3, -1234567
li $4, -126
mult $3, $4
mfhi $5
mfhi $6
multu $3, $4
mfhi $5
mfhi $6
div $3, $4
mfhi $5
mfhi $6
divu $3, $4
mfhi $5
mfhi $6
```

预期输出如下：

```
@00003000: $ 1 <= 00120000
@00003004: $ 3 <= 0012d687
@00003008: $ 1 <= 00250000
@0000300c: $ 4 <= 0025ad0f
@00003014: $ 5 <= 000002c5
@00003018: $ 6 <= 000002c5
@00003020: $ 5 <= 000002c5
@00003024: $ 6 <= 000002c5
@0000302c: $ 5 <= 0012d687
@00003030: $ 6 <= 0012d687
@00003038: $ 5 <= 0012d687
@0000303c: $ 6 <= 0012d687
@00003048: $ 1 <= ffed0000
@0000304c: $ 3 <= ffed2979
@00003050: $ 4 <= 0000007e
@00003058: $ 5 <= ffffffff
@0000305c: $ 6 <= ffffffff
@00003064: $ 5 <= 0000007d
```

```
@00003068: $ 6 <= 0000007d
@00003070: $ 5 <= fffffffed
@00003074: $ 6 <= fffffffed
@0000307c: $ 5 <= 0000006f
@00003080: $ 6 <= 0000006f
@00003084: $ 1 <= 00120000
@00003088: $ 3 <= 0012d687
@0000308c: $ 4 <= ffffff82
@00003094: $ 5 <= ffffffff
@00003098: $ 6 <= ffffffff
@000030a0: $ 5 <= 0012d686
@000030a4: $ 6 <= 0012d686
@000030ac: $ 5 <= 00000013
@000030b0: $ 6 <= 00000013
@000030b8: $ 5 <= 0012d687
@000030bc: $ 6 <= 0012d687
@000030c0: $ 1 <= ffed0000
@000030c4: $ 3 <= ffed2979
@000030c8: $ 4 <= ffffff82
@000030d0: $ 5 <= 00000000
@000030d4: $ 6 <= 00000000
@000030dc: $ 5 <= ffed28fb
@000030e0: $ 6 <= ffed28fb
@000030e8: $ 5 <= fffffffed
@000030ec: $ 6 <= fffffffed
@000030f4: $ 5 <= ffed2979
@000030f8: $ 6 <= ffed2979
```

其次，使用自动生成程序生成大量测试代码：

生成的部分代码示例

```
INTI0: li $gp, 0x1800
INTI1: li $sp, 0x2fff
INTI2: li $10, 13
INTI3: li $11, 0
INTI4: li $12, -12
L0: sllv $5, $2, $3
L1: multu $3, $0
L2: jal L3
L3: divu $5, $sp
L4: slti $31, $5, -7217
L5: div $31, $sp
L6: addiu $4, $31, 8165
L7: lh $2, 0xffc($0)
L8: sh $3, 746($0)
```

```
L9: bne $11, $10, L11
L10: lui $3, 18388
L11: srlv $5, $0, $2
L12: div $2, $sp
L13: addu $31, $3, $31
L14: andi $5, $31, 35112
L15: subu $3, $2, $31
L16: sw $31, 4092($0)
L17: sb $31, 775($0)
L18: xor $4, $31, $31
L19: addu $3, $0, $0
L20: lui $4, 20445
L21: mfhi $2
L22: mult $2, $3
L23: sltu $4, $3, $31
L24: subu $5, $0, $3
L25: and $4, $3, $2
L26: sllv $4, $0, $0
L27: slt $5, $3, $4
L28: multu $5, $0
L29: addiu $2, $2, -24700
L30: sllv $31, $2, $3
L31: lui $2, 12554
L32: mfhi $3
L33: sltu $5, $2, $0
L34: lbu $5, 0xfff($0)
L35: blez $0, L38
L36: multu $4, $0
L37: or $2, $31, $0
L38: mult $5, $0
L39: div $31, $sp
L40: j L43
L41: addu $3, $0, $0
L42: mtlo $0
L43: mthi $4
L44: nor $3, $4, $0
L45: beq $10, $12, L49
L46: slti $3, $2, 17074
L47: ori $5, $5, 59903
L48: and $3, $31, $2
L49: divu $3, $sp
L50: sltiu $31, $31, 7711
L51: mult $3, $3
L52: slti $3, $2, -29153
L53: xori $4, $2, 24694
```

```
L54: multu $4, $3
L55: sw $31, 2768($0)
L56: sw $5, 3292($0)
L57: xori $31, $3, 48165
L58: or $4, $2, $3
L59: and $3, $0, $0
L60: subu $3, $3, $3
L61: xori $31, $3, 53382
L62: sw $2, 2076($0)
L63: xor $3, $3, $4
L64: and $4, $0, $0
L65: mthi $31
L66: lb $31, 0x3($0)
L67: lb $4, 0xddf($0)
L68: lui $4, 25698
L69: sltiu $2, $3, 3110
L70: xor $0, $0, $31
L71: multu $2, $31
L72: srlv $4, $0, $0
L73: mtlo $3
L74: lui $5, 32765
L75: andi $3, $4, 25681
L76: sh $4, 2($0)
L77: sllv $2, $4, $0
L78: sltu $3, $3, $31
L79: jal L81
L80: mult $3, $3
L81: and $2, $31, $4
L82: bltz $3, L84
L83: mthi $2
L84: slt $0, $3, $31
L85: lh $3, 0x9b4($0)
L86: andi $4, $5, 23911
L87: sb $0, 2146($0)
L88: ori $3, $5, 56061
L89: xor $5, $2, $4
L90: sw $2, 4092($0)
L91: lhu $2, 0xe66($0)
L92: slt $31, $2, $3
L93: multu $3, $3
L94: srav $4, $31, $0
L95: bgez $31, L99
L96: mfhi $31
L97: bgez $31, L100
L98: mfhi $2
```

```
L99: sw $31, 0($0)
L100: sw $3, 3376($0)
L101: lh $3, 0x288($0)
L102: mult $4, $4
L103: ori $0, $31, 31393
L104: sw $5, 4092($0)
L105: mtlo $31
L106: srlv $3, $31, $0
L107: slti $2, $4, -22251
L108: divu $2, $sp
L109: blez $2, L113
L110: lh $5, 0xffc($0)
```

构造策略：主要只使用 0，2，3，4，31 号寄存器，减少对 0 号寄存器的写入，提高冒险发生的可能性。10，11，12 号寄存器存放用来比较的特定值，跳转指令只向下跳避免死循环，同样 jr 暂不测试留待手动测试。beq 增大相同寄存器概率，增大条件成立可能。

## 五、思考题

**1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？**

乘除法电路复杂，如果要求在一个时钟周期内完成，会严重拖慢整体时钟周期。将乘除法部件独立出来，允许乘除法运算经过多个周期才得出结果，可以加快系统整体时钟速度，符合加速大概率事件思想。因此，需要独立的，不会每周期自动更新导致值丢失的寄存器来存储算出的值，因此有独立的 Hi/Lo。此外，32 位乘除结果可能达到 64 位，需要独立的 Hi，Lo 寄存器存储。

**2. 参照你对延迟槽的理解，试解释“乘除槽”。**

乘除法进行时，会有 busy 信号，此时其他非乘除法指令仍会执行，但乘除法指令不能执行会暂停。且影响不止一条指令。

**3. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）**

由于内存性能瓶颈一般在于响应速度方面而数据传输量不同带来的速度差异不是那么大，如果连续访问的话，按字访问并读取显然性能更有优势。但是如果访问不连续且访问间隔大于一次数据读写的数据，此时按字节访问就更具有优势。

**5. 如何概括你所设计的 CPU 的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？**

侦测者型。采用大量宏定义和冗余流水线寄存器位，以保证每级流水线的在结构上尽可能相似，减少修改。添加语法意义上冗余的说明符号，来帮助不产生歧义。模块命名规范会导致变量名长且复杂化，但更加直观。利用专用 MUX 模块，虽然导致了耦合性增加但整体而言减少了出错可能性。

**6. 你对流水线 CPU 设计风格有何见解？**

我认为，鉴于目前课上测试时间紧难度大，在设计时我试图以课下编码的复杂性和长耗时换取设计的正确性以及课上增添指令的便捷性。因此，在本人看来，侦测者风格更加适合。同时，因为添加了大量冗余，在增加指令时就不会束手无策。同时，正常代码足够规整，方可支撑课上出现困难时的 trick。

**7. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？**

本人通过自动生成测试数据解决测试问题，详细信息可以参见前文。主要问题在于新添加的 mdu 部件相关指令。其中之一就是两条连续 mdu 有关指令的暂停（被坑到了耶（x）。

同样，jalr 和 jr 需要手动测试，但是其转发措施和 beq 等指令有类似之处，也就不难处理。