# 计算机组成原理P5实验报告

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为Verilog实现的流水线MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、j、jal、jr、nop}。为了实现这些功能，CPU在顶层模块mips下并列包含了DATAPATH、FORWARD\_CONTROL、STOP\_CONTROL三个模块。

DATAPATH模块下分五个流水级F、D、E、M、W。F级包含了PC、IM部件和用于PC选择的MUX\_PcSel、用于pc+4的add4两个小部件，D级包含了GRF、EXT、NPC、CMP部件和用于写入寄存器A3选择的功能性部件MUX\_RegA3Sel；E级包含了ALU部件和用于ALUB选择的功能性部件MUX\_AluBSel；M级包含了DM部件；W级连接到D级寄存器，包含了用于存入寄存器数据选择的功能性部件MUX\_RegDataSel。相邻两流水级之间还各设置了一个流水线寄存器regD、regE、regM、regW，用于存储流水的信息。此外，为满足数据冒险的转发需求，还设置了五个转发多路选择器：D级MFCMP1D、MFCMP2D分别用于选择需进入CMP部件和E级寄存器的两个寄存器值；E级MFALUAE、用于选择参与ALU运算的第一个数据，MFALUBE用于选择参与ALU运算的第二个来自寄存器的数据和进入M级寄存器的数据；MFDM用于选择写入DM的数据。

FORWARD\_CONTROL模块用于生成转发信号控制转发。

STOP\_CONTROL模块用于生成暂停信号控制暂停。

此外，工程文件中包含了名为define的.v文件用于宏定义。本流水线CPU主要采用分布式译码，主要将指令与A3流水，所有需要指令信息的部件下均包括了译码部件DECODE，用于产生改位置指令操作所需信息。

### 关键模块定义

1. DATAPATH
2. PC

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位至0x00003000） |
| en | I | 使能信号，为1时PC正常工作，为0时PC冻结 |
| pcin[31:0] | I | 32位输入次指令地址 |
| pc[31:0] | O | 32位输出当前指令地址 |

1. IM

|  |  |  |
| --- | --- | --- |
| 功能描述 | 指令存储器，内含32\*1024字存储器，根据输入的地址输出指令 | |
| 信号名 | 方向 | 描述 |
| pc[31:0] | I | 32位输入当前指令地址 |
| instr[31:0] | O | 32位输出指令信号 |

1. GRF

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| instr[31:0] | I | 31位输入D级指令，译码出写使能 |
| instr\_W[31:0] | I | 31位输入W级指令，译码出写使能 |
| A3\_W[4:0] | I | 写入寄存器编号 |
| RegData[31:0] | I | 回写寄存器的值 |
| pc4[31:0] | I | 32位输入当前pc+4，用于display |
| RD1[31:0] | O | 第一个寄存器编号读出的值 |
| RD2[31:0] | O | 第二个寄存器编号读出的值 |

1. CMP

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于前置的beq跳转判断 | |
| 信号名 | 方向 | 描述 |
| D1[31:0] | I | 32位输入用于判断的第一个数据 |
| D2[31:0] | I | 32位输入用于判断的第一个数据 |
| zero | O | 1位输出判断结果，为0表示不相等，1表示相等 |

1. EXT

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能 |
| instr[31:0] | I | 32为输入当前指令，译码出操作选择信号和用于操作的立即数 |
| ext[31:0] | O | 32位输出扩展结果 |

1. NPC

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc4[31:0] | I | 32位输入当前pc+4 |
| instr[31:0] | I | 32位输入当前指令，译码出操作选择信号 |
| zero | I | 辅助选择NPC（目前仅为beq使用） |
| ra[31:0] | I | 32位输入来自寄存器的指令地址 |
| npc[31:0] | O | 32位输出用于跳转的次指令地址 |

1. ALU

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 第一个32位操作数 |
| B[31:0] | I | 第二个32位操作数 |
| instr[31:0] | I | 32位输入当前指令，译码出操作选择信号 |
| C[31:0] | O | 32位输出计算结果 |

1. DM

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| pc4[31:0] | I | 32位输入当前pc+4 |
| instr[31:0] | I | 32位输入当前指令 |
| Addr[31:0] | I | 32位输入写入DM的地址 |
| Data[31:0] | I | 32位输入写入的数据 |
| DMRD[31:0] | O | 32位输出读出的数据 |

1. regD

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| instr[31:0] | I | 32位输入当前指令 |
| pc4[31:0] | I | 32位输入当前pc+4 |
| en | I | 使能信号，1则寄存器正常工作，0则冻结寄存器所存内容 |
| instr\_D[31:0] | O | 32位输出D级指令 |
| pc4\_D[31:0] | O | 32位输出D级pc+4 |

1. regE

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| clr | I | 清空信号，1则清除寄存器中所有值，0则寄存器正常工作 |
| instr[31:0] | I | 32位输入当前指令 |
| V1[31:0] | I | 32位输入经转发选择后的rs寄存器值 |
| V2[31:0] | I | 32位输入经转发选择后的rt寄存器值 |
| ext[31:0] | I | 32位输入ext扩展结果 |
| pc4[31:0] | I | 32位输入当前pc+4 |
| A3[4:0] | I | 5位输入当前指令对应写入寄存器编号 |
| instr\_E[31:0] | O | 32位输出E级指令 |
| V1\_E[31:0] | O | 32位输出E级V1 |
| V2\_E[31:0] | O | 32位输出E级V2 |
| ext\_E[31:0] | O | 32位输出E级ext |
| pc4\_E[31:0] | O | 32位输出E级pc+4 |
| A3\_E[4:0] | O | 5位输出E级指令对应的写入寄存器编号 |

1. regM

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| instr[31:0] | I | 32位输入当前指令 |
| V2[31:0] | I | 32位输入V2 |
| ALUC[31:0] | I | 32位输入ALU计算结果 |
| pc4[31:0] | I | 32位输入当前pc+4 |
| A3[4:0] | I | 5位输入当前指令对应写入寄存器编号 |
| instr\_M[31:0] | O | 32位输出M级指令 |
| V2\_M[31:0] | O | 32位输出M级V2 |
| ALUC\_M[31:0] | O | 32位输出M级ALU计算结果 |
| pc4\_M[31:0] | O | 32位输出M级pc+4 |
| A3\_M[4:0] | O | 5位输出M级指令对应的写入寄存器编号 |

1. regW

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| instr[31:0] | I | 32位输入当前指令 |
| pc4[31:0] | I | 32位输入当前pc+4 |
| ALUC[31:0] | I | 32位输入ALU计算结果 |
| DMRD[31:0] | I | 32位输入DM读取结果 |
| A3[4:0] | I | 5位输入当前指令对应写入寄存器编号 |
| instr\_W[31:0] | O | 32位输出W级指令 |
| pc4\_W[31:0] | O | 32位输出W级pc+4 |
| ALUC\_W[31:0] | O | 32位输出W级ALU计算结果 |
| DMRD\_W[31:0] | O | 32位输出W级DM读取结果 |
| A3\_W[4:0] | O | 5位输出W级指令对应的写入寄存器编号 |

1. add4

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于pc+4 | |
| 信号名 | 方向 | 描述 |
| pc[31:0] | I | 32位输入当前指令地址 |
| pc4[31:0] | O | 32位输出pc+4 |

1. MUX\_PcSel

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择使用普通pc+4或跳转指令地址 | |
| 信号名 | 方向 | 描述 |
| pc4[31:0] | I | 32位输入pc+4 |
| instr[31:0] | I | 32位输入当前指令，译码出选择信号 |
| npc[31:0] | I | 32位输入当前指令，译码用于选择进入pc的次指令地址 |
| pcin[31:0] | O | 32位输出进入pc的次指令地址 |

1. MUX\_RegA3Sel

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择写入寄存器编号 | |
| 信号名 | 方向 | 描述 |
| instr[31:0] | I | 32位输入当前指令，译码出选择信号 |
| A3[4:0] | O | 5位输出当前指令对应写入寄存器编号 |

1. MUX\_AluBSel

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择进入ALU计算的B数据 | |
| 信号名 | 方向 | 描述 |
| instr[31:0] | I | 32位输入当前指令，译码出选择信号 |
| V2[31:0] | I | 32位输入经转发选择后的rt寄存器数据 |
| ext[31:0] | I | 32位输入ext计算结果 |
| ALUB[31:0] | O | 32位输出进入ALU的B数据 |

1. MUX\_RegDataSel

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择存入寄存器的数据 | |
| 信号名 | 方向 | 描述 |
| instr[31:0] | I | 32位输入当前指令，译码出选择信号 |
| ALUC[31:0] | I | 32位输入ALU计算结果 |
| DMRD[31:0] | I | 32位输入DM读出数据 |
| pc4[31:0] | I | 32位输入当前pc+4 |
| RegData[31:0] | O | 32位输出存入寄存器数据 |

1. MFCMP1D

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择转发来的rs数据进入CMP和E级寄存器 | |
| 信号名 | 方向 | 描述 |
| mfcmp1dSel[3:0] | I | 4位输入转发选择信号 |
| RD1[31:0] | I | 32位输入当前位置指令从寄存器读取的数据 |
| ALUC\_M[31:0] | I | 32位输入来自M级的ALU计算结果 |
| pc4\_M[31:0] | I | 32位输入来自M级的pc4 |
| ALUC\_W[31:0] | I | 32位输入来自W级的ALU计算结果 |
| DMRD\_W[31:0] | I | 32位输入来自W级的DM读取结果 |
| pc4\_W[31:0] | I | 32位输入来自W级的pc4 |
| mfcmp1[31:0] | O | 32位输出选择出的当前指令的真正rs读取数据 |

1. MFCMP2D

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择转发来的rt数据进入CMP和E级寄存器 | |
| 信号名 | 方向 | 描述 |
| mfcmp2dSel[3:0] | I | 4位输入转发选择信号 |
| RD2[31:0] | I | 32位输入当前位置指令从寄存器读取的数据 |
| ALUC\_M[31:0] | I | 32位输入来自M级的ALU计算结果 |
| pc4\_M[31:0] | I | 32位输入来自M级的pc4 |
| ALUC\_W[31:0] | I | 32位输入来自W级的ALU计算结果 |
| DMRD\_W[31:0] | I | 32位输入来自W级的DM读取结果 |
| pc4\_W[31:0] | I | 32位输入来自W级的pc4 |
| mfcmp2[31:0] | O | 32位输出选择出的当前指令的真正rt读取数据 |

1. MFALUAE

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择转发来的真正寄存器值进入ALU作为A | |
| 信号名 | 方向 | 描述 |
| mfaluaeSel[3:0] | I | 4位输入转发选择信号 |
| V1\_E[31:0] | I | 32位输入来自E级寄存器流水的V1值 |
| ALUC\_M[31:0] | I | 32位输入来自M级的ALU计算结果 |
| pc4\_M[31:0] | I | 32位输入来自M级的pc4 |
| ALUC\_W[31:0] | I | 32位输入来自W级的ALU计算结果 |
| DMRD\_W[31:0] | I | 32位输入来自W级的DM读取结果 |
| pc4\_W[31:0] | I | 32位输入来自W级的pc4 |
| mfalua[31:0] | O | 32位输出选择出的当前指令进入ALU的真正寄存器值 |

1. MFALUBE

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择转发来的真正寄存器值进入ALU的B值选择 | |
| 信号名 | 方向 | 描述 |
| mfalubeSel[3:0] | I | 4位输入转发选择信号 |
| V2\_E[31:0] | I | 32位输入来自E级寄存器流水的V2值 |
| ALUC\_M[31:0] | I | 32位输入来自M级的ALU计算结果 |
| pc4\_M[31:0] | I | 32位输入来自M级的pc4 |
| ALUC\_W[31:0] | I | 32位输入来自W级的ALU计算结果 |
| DMRD\_W[31:0] | I | 32位输入来自W级的DM读取结果 |
| pc4\_W[31:0] | I | 32位输入来自W级的pc4 |
| mfalub[31:0] | O | 32位输出选择出的当前指令用于ALUB选择的真正寄存器值 |

1. MFDM

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于选择转发来的真正数据用于写入DM | |
| 信号名 | 方向 | 描述 |
| mfdmSel[3:0] | I | 4位输入转发选择信号 |
| V2\_M[31:0] | I | 32位输入来自M级寄存器流水的V2值 |
| ALUC\_W[31:0] | I | 32位输入来自W级的ALU计算结果 |
| pc4\_W[31:0] | I | 32位输入来自W级的pc4 |
| DMRD\_W[31:0] | I | 32位输入来自W级的DM读取结果 |
| mfdm[31:0] | O | 32位输出选择出的当前指令用于存入DM的真正数据 |

1. FORWARD\_CONTROL

|  |  |  |
| --- | --- | --- |
| 功能描述 | 立足D级，用于转发控制信号的生成 | |
| 信号名 | 方向 | 描述 |
| instr\_D[31:0] | I | 32位输入位于D级的指令 |
| instr\_E[31:0] | I | 32位输入位于E级的指令 |
| instr\_M[31:0] | I | 32位输入位于M级的指令 |
| instr\_W[31:0] | I | 32位输入位于W级的指令 |
| A3\_M[4:0] | I | 5位输入M级指令的写入寄存器地址 |
| A3\_W[4:0] | I | 5位输入W级指令的写入寄存器地址 |
| mfcmp1dSel[3:0] | O | 4位输出cmp1转发数据选择信号 |
| mfcmp2dSel[3:0] | O | 4位输出cmp2转发数据选择信号 |
| mfaluaeSel[3:0] | O | 4位输出ALUA转发数据选择信号 |
| mfalubeSel[3:0] | O | 4位输出ALUB转发数据选择信号 |
| mfdmSel[3:0] | O | 4位输出DM转发数据选择信号 |

1. STOP\_CONTROL

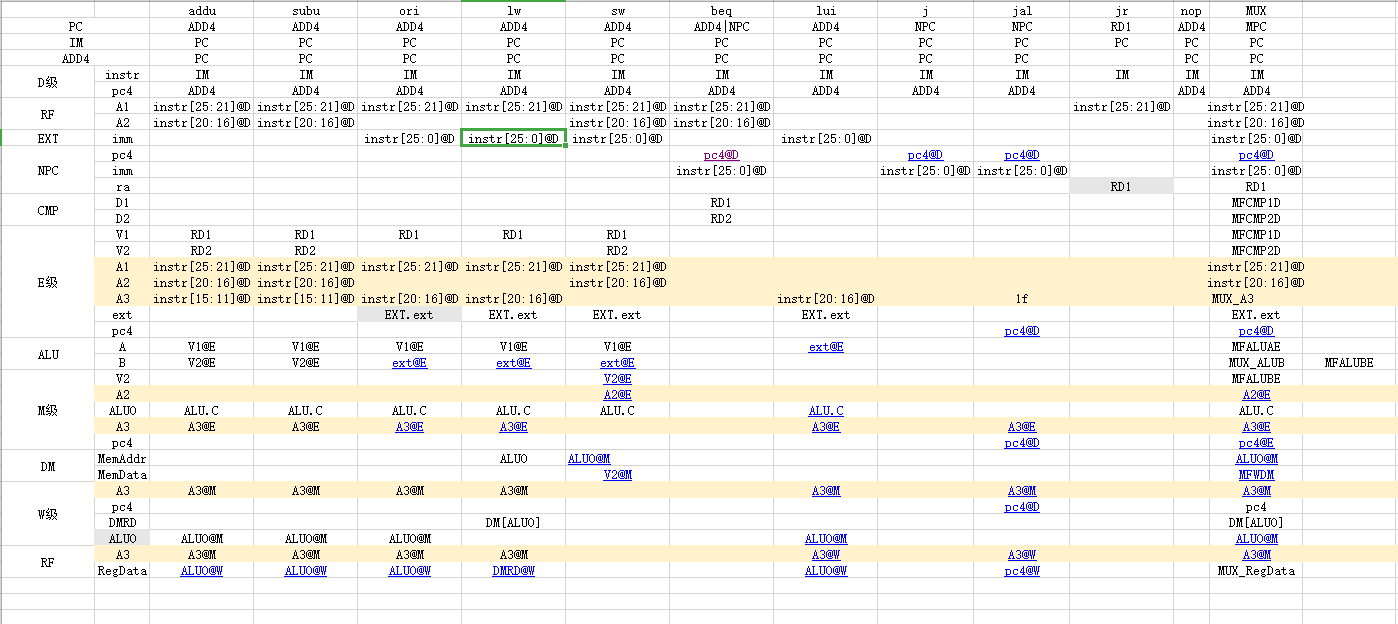
|  |  |  |
| --- | --- | --- |
| 功能描述 | 放眼全局，用于暂停控制信号的生成 | |
| 信号名 | 方向 | 描述 |
| instr\_D[31:0] | I | 32位输入位于D级的指令 |
| instr\_E[31:0] | I | 32位输入位于E级的指令 |
| instr\_M[31:0] | I | 32位输入位于M级的指令 |
| A3\_E[4:0] | I | 5位输入E级指令的写入寄存器地址 |
| A3\_M[4:0] | I | 5位输入M级指令的写入寄存器地址 |
| enPC | O | 输出用于暂停的PC使能信号 |
| enD | O | 输出用于暂停的D级流水线寄存器使能信号 |
| clrE | O | 输出用于暂停的E级流水线寄存器清空信号 |

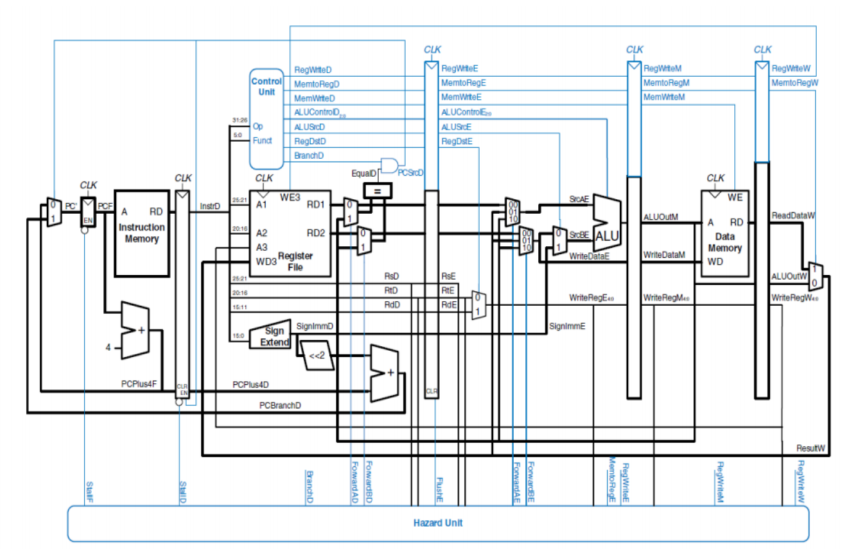
1. DECODE

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于译码产生各种控制信号 | |
| 信号名 | 方向 | 描述 |
| instr[31:0] | I | 32位输入指令信号 |
| NPCOp[2:0] | O | 3位输出npc操作选择信号 |
| EXTOp[3:0] | O | 4位输出ext操作选择信号 |
| ALUOp[3:0] | O | 4位输出ALU操作选择信号 |
| RegWrite | O | 1位输出寄存器写使能信号 |
| MemWrite | O | 1位输出DM写使能信号 |
| RegA3Sel[2:0] | O | 3位输出写入寄存器编号选择信号 |
| RegDataSel[2:0] | O | 3位输出写入寄存器数据选择信号 |
| AluBSel[2:0] | O | 3位输出ALUB选择信号 |
| PcSel[1:0] | O | 2位输出pc次指令地址选择信号 |
| Tuse\_rs0 | O | 表示指令经0周期使用rs寄存器值 |
| Tuse\_rs1 | O | 表示指令经1周期使用rs寄存器值 |
| Tuse\_rt0 | O | 表示指令经0周期使用rt寄存器值 |
| Tuse\_rt1 | O | 表示指令经1周期使用rt寄存器值 |
| Tuse\_rt2 | O | 表示指令经2周期使用rt寄存器值 |
| Tnew[2:0] | O | 表示指令产生写入寄存器值的部件 |

### 数据通路的综合

1. 所有指令的指令级别数据通路





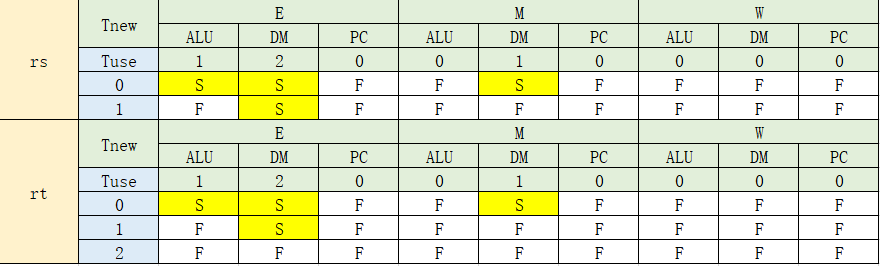
### （四）重要机制实现方法

1. 跳转

NPC模块译码出NPC的计算控制信号，计算出npc后输出到F级MUX\_PcSel，MUX\_PcSel对来自D级寄存器的指令（即与NPC同指令）译码判断是否为跳转指令，若是则输出npc至PC；反之输出当前F级pc+4（来自add4部件的输出）至PC。

1. 暂停





构建策略矩阵，当Tnew>Tuse时必须使用暂停，产生暂停信号，此时需要冻结PC的值，冻结D级流水线的值并清零E级流水线。

暂停信号由rs、rt寄存器的暂停信号取并集，当Tnew与Tuse满足矩阵中暂停条件，且写入寄存器与读取寄存器相同时相应暂停信号置1。

对D、E、M级指令分别译码即可得到Tuse与Tnew值。

1. 转发

沿用Tnew信号（此处似乎与教程和课件内容不甚相符），本设计在宏定义时不同Tnew信号反映该级目前指令产生写入寄存器值的部件，因此在转发控制中只需要根据该级处指令写入寄存器编号与D级指令读取寄存器编号相同、该级指令产生写入寄存器值的部件两个信号判断转发哪一数据。

对M、W级指令分别译码即可得到该两级产生的写入寄存器数据来源，从而明确转发的数据来源。

此处需注意，如果写入寄存器为0号寄存器，不进行转发。

## 测试方案

### 典型测试样例

|  |  |
| --- | --- |
| Tuse\_rs=0，Tnew\_E=ALU，暂停 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x302c ##  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003014: $10 <= 00000005  @0000301c: $12 <= 0000302c  @00003024: $ 1 <= 00000009  @0000302c: $12 <= 00000146 |
| Tuse\_rs=0，Tnew\_E=DM，暂停 | |
| ori $1, $0, 256  ori $6, $0, 256  ori $5, $0, 4  sw $1, 4($5)  lw $10, 4($5) ##  beq $10, $6, label ##  nop  ori $20, $0, 2  label:  lui $4, 0x1234 | @00003000: $ 1 <= 00000100  @00003004: $ 6 <= 00000100  @00003008: $ 5 <= 00000004  @0000300c: \*00000008 <= 00000100  @00003010: $10 <= 00000100  @00003020: $ 4 <= 12340000 |
| Tuse\_rs=0，Tnew\_E=PC（似乎这样的搭配只有延迟槽里跳转的情况） | |
| Tuse\_rs=0，Tnew\_M=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $30, $0, 56  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x3034 ##  ori $29, $0, 58  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $30 <= 00000038  @00003018: $10 <= 00000005  @00003020: $12 <= 00003034  @00003024: $29 <= 0000003a  @0000302c: $ 1 <= 00000009  @00003034: $12 <= 00000146 |
| Tuse\_rs=0，Tnew\_M=DM，暂停 | |
| ori $1, $0, 0x3038  ori $5, $0, 0x3038  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $3, $0, 123  beq $10, $5, label ##  nop  ori $25, $0, 256  label:  lw $6, -4($2) ##  lui $26, 0x1234  jr $6 ##  nop  ori $23, $0, 125  ori $24, $0, 156 | @00003000: $ 1 <= 00003038  @00003004: $ 5 <= 00003038  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003038  @00003010: $10 <= 00003038  @00003014: $ 3 <= 0000007b  @00003024: $ 6 <= 00003038  @00003028: $26 <= 12340000  @00003038: $24 <= 0000009c |
| Tuse\_rs=0，Tnew\_M=PC，转发 | |
| ori $5, $0, 0x3010  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  beq $31, $5, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003010  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @00003020: $ 9 <= 12340000 |
| Tuse\_rs=0，Tnew\_W=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $27, $0, 256  ori $22, $0, 355  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x303c ##  addu $23, $2, $4  subu $24, $23, $4  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $27 <= 00000100  @00003014: $22 <= 00000163  @0000301c: $10 <= 00000005  @00003024: $12 <= 0000303c  @00003028: $23 <= 00003001  @0000302c: $24 <= 00003000  @00003034: $ 1 <= 00000009  @0000303c: $12 <= 00000146 |
| Tuse\_rs=0，Tnew\_W=DM，转发 | |
| ori $1, $0, 0x3040  ori $5, $0, 0x3040  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $16, $0, 258  ori $3, $0, 123  beq $10, $5, label ##  nop  ori $25, $0, 256  label:  lw $6, -4($2) ##  lui $26, 0x1234  addu $17, $16, $3  jr $6 ##  nop  ori $23, $0, 125  ori $24, $0, 156 | @00003000: $ 1 <= 00003040  @00003004: $ 5 <= 00003040  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003040  @00003010: $10 <= 00003040  @00003014: $16 <= 00000102  @00003018: $ 3 <= 0000007b  @00003028: $ 6 <= 00003040  @0000302c: $26 <= 12340000  @00003030: $17 <= 0000017d  @00003040: $24 <= 0000009c |
| Tuse\_rs=0，Tnew\_W=PC，转发 | |
| ori $5, $0, 0x3010  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  ori $10, $0, 0x1234  beq $31, $5, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003010  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @00003014: $10 <= 00001234  @00003024: $ 9 <= 12340000 |
| Tuse\_rs=1，Tnew\_E=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  addu $4, $2, $1 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 4 <= 000001ff |
| Tuse\_rs=1，Tnew\_E=DM，暂停 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  addu $4, $3, $1 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $ 4 <= 00001238 |
| Tuse\_rs=1，Tnew\_E=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $3, $31, $1 ##  addu $4, $1, $2  label:  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 3 <= 000030aa  @00003014: $ 5 <= 00000093 |
| Tuse\_rs=1，Tnew\_M=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  ori $3, $0, 289  addu $4, $2, $1 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 3 <= 00000121  @0000300c: $ 4 <= 000001ff |
| Tuse\_rs=1，Tnew\_M=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  ori $25, $0, 156  addu $4, $3, $1 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $25 <= 0000009c  @00003014: $ 4 <= 00001238 |
| Tuse\_rs=1，Tnew\_M=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $4, $1, $2  lui $8, 0x1235  label:  addu $3, $31, $1 ##  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 4 <= 00000109  @00003014: $ 3 <= 000030aa  @00003018: $ 5 <= 00000093 |
| Tuse\_rs=1，Tnew\_W=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  lui $5, 0x1256  ori $3, $0, 289  addu $4, $2, $1 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 5 <= 12560000  @0000300c: $ 3 <= 00000121  @00003010: $ 4 <= 000001ff |
| Tuse\_rs=1，Tnew\_W=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  ori $25, $0, 156  lui $26, 0x1475  addu $4, $3, $1 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $25 <= 0000009c  @00003014: $26 <= 14750000  @00003018: $ 4 <= 00001238 |
| Tuse\_rs=1，Tnew\_W=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $4, $1, $2  lui $8, 0x1235  label:  lui $9, 0x58  addu $3, $31, $1 ##  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 4 <= 00000109  @00003014: $ 9 <= 00580000  @00003018: $ 3 <= 000030aa  @0000301c: $ 5 <= 00000093 |

|  |  |
| --- | --- |
| Tuse\_rt=0，Tnew\_E=ALU，暂停 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  beq $2, $5, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x302c | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003014: $10 <= 00000005  @0000301c: $12 <= 0000302c |
| Tuse\_rt=0，Tnew\_E=ALU，暂停 | |
| ori $1, $0, 256  ori $6, $0, 256  ori $5, $0, 4  sw $1, 4($5)  lw $10, 4($5) ##  beq $6, $10, label ##  nop  ori $20, $0, 2  label:  lui $4, 0x1234 | @00003000: $ 1 <= 00000100  @00003004: $ 6 <= 00000100  @00003008: $ 5 <= 00000004  @0000300c: \*00000008 <= 00000100  @00003010: $10 <= 00000100  @00003020: $ 4 <= 12340000 |
| Tuse\_rt=0，Tnew\_E=PC（似乎这样的搭配只有延迟槽里跳转的情况） | |
| Tuse\_rt=0，Tnew\_M=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $30, $0, 56  beq $2, $5, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x3034 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $30 <= 00000038  @00003018: $10 <= 00000005  @00003020: $12 <= 00003034 |
| Tuse\_rt=0，Tnew\_M=DM，暂停 | |
| ori $1, $0, 0x3038  ori $5, $0, 0x3038  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $3, $0, 123  beq $5, $10, label ##  nop  ori $25, $0, 256  label:  ori $27, $0, 145 | @00003000: $ 1 <= 00003038  @00003004: $ 5 <= 00003038  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003038  @00003010: $10 <= 00003038  @00003014: $ 3 <= 0000007b  @00003024: $27 <= 00000091 |
| Tuse\_rt=0，Tnew\_M=PC，转发 | |
| ori $5, $0, 0x3011  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  beq $5, $31, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003011  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @0000301c: $ 6 <= 000000f7  @00003020: $ 9 <= 12340000 |
| Tuse\_rt=0，Tnew\_W=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $27, $0, 256  ori $22, $0, 355  beq $2, $5, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  addu $23, $2, $4 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $27 <= 00000100  @00003014: $22 <= 00000163  @0000301c: $10 <= 00000005  @00003024: $23 <= 00003001 |
| Tuse\_rt=0，Tnew\_W=DM，转发 | |
| ori $1, $0, 0x3040  ori $5, $0, 0x3040  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $16, $0, 258  ori $3, $0, 123  beq $5, $10, label ##  nop  ori $25, $0, 256  label:  lw $6, -4($2) | @00003000: $ 1 <= 00003040  @00003004: $ 5 <= 00003040  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003040  @00003010: $10 <= 00003040  @00003014: $16 <= 00000102  @00003018: $ 3 <= 0000007b  @00003028: $ 6 <= 00003040 |
| Tuse\_rt=0，Tnew\_W=PC，转发 | |
| ori $5, $0, 0x3010  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  ori $10, $0, 0x1234  beq $5, $31, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003010  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @00003014: $10 <= 00001234  @00003024: $ 9 <= 12340000 |
| Tuse\_rt=1，Tnew\_E=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  addu $4, $1, $2 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 4 <= 000001ff |
| Tuse\_rt=1，Tnew\_E=DM，暂停 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  addu $4, $1, $3 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $ 4 <= 00001238 |
| Tuse\_rt=1，Tnew\_E=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $3, $1, $31 ##  addu $4, $1, $2  label:  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 3 <= 000030aa  @00003014: $ 5 <= 00000093 |
| Tuse\_rt=1，Tnew\_M=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  ori $3, $0, 289  addu $4, $1, $2 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 3 <= 00000121  @0000300c: $ 4 <= 000001ff |
| Tuse\_rt=1，Tnew\_M=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  ori $25, $0, 156  addu $4, $1, $3 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $25 <= 0000009c  @00003014: $ 4 <= 00001238 |
| Tuse\_rt=1，Tnew\_M=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $4, $1, $2  lui $8, 0x1235  label:  addu $3, $31, $1 ##  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 4 <= 00000109  @00003014: $ 3 <= 000030aa  @00003018: $ 5 <= 00000093 |
| Tuse\_rt=1，Tnew\_W=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  lui $5, 0x1256  ori $3, $0, 289  addu $4, $1, $2 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 5 <= 12560000  @0000300c: $ 3 <= 00000121  @00003010: $ 4 <= 000001ff |
| Tuse\_rt=1，Tnew\_W=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  ori $25, $0, 156  lui $26, 0x1475  addu $4, $1, $3 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $25 <= 0000009c  @00003014: $26 <= 14750000  @00003018: $ 4 <= 00001238 |
| Tuse\_rt=1，Tnew\_W=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $4, $1, $2  lui $8, 0x1235  label:  lui $9, 0x58  addu $3, $1, $31 ##  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 4 <= 00000109  @00003014: $ 9 <= 00580000  @00003018: $ 3 <= 000030aa  @0000301c: $ 5 <= 00000093 |
| Tuse\_rt=2，Tnew\_E=ALU，转发 | |
| ori $1, $0, 4  ori $5, $0, 256  addu $3, $1, $1 ##  sw $3, 4($5) ## | @00003000: $ 1 <= 00000004  @00003004: $ 5 <= 00000100  @00003008: $ 3 <= 00000008  @0000300c: \*00000104 <= 00000008 |
| Tuse\_rt=2，Tnew\_E=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1243  sw $2, -4($1)  lw $3, -4($1) ##  sw $3, 8($1) ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001243  @00003008: \*00000000 <= 00001243  @0000300c: $ 3 <= 00001243  @00003010: \*0000000c <= 00001243 |
| Tuse\_rt=2，Tnew\_E=PC，转发 | |
| ori $1, $0, 4  jal label ##  sw $31, 4($1) ##  addu $3, $1, $1  label:  ori $5, $0, 156 | @00003000: $ 1 <= 00000004  @00003004: $31 <= 0000300c  @00003008: \*00000008 <= 0000300c  @00003010: $ 5 <= 0000009c |
| Tuse\_rt=2，Tnew\_M=ALU，转发 | |
| ori $1, $0, 4  ori $5, $0, 256  addu $3, $1, $1 ##  ori $4, $0, 28  sw $3, 4($5) ## | @00003000: $ 1 <= 00000004  @00003004: $ 5 <= 00000100  @00003008: $ 3 <= 00000008  @0000300c: $ 4 <= 0000001c  @00003010: \*00000104 <= 00000008 |
| Tuse\_rt=2，Tnew\_M=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1243  sw $2, -4($1)  lw $3, -4($1) ##  ori $9, $0, 156  sw $3, 8($1) ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001243  @00003008: \*00000000 <= 00001243  @0000300c: $ 3 <= 00001243  @00003010: $ 9 <= 0000009c  @00003014: \*0000000c <= 00001243 |
| Tuse\_rt=2，Tnew\_M=PC，转发 | |
| ori $1, $0, 4  jal label ##  nop  addu $3, $1, $1  label:  sw $31, 4($1) ##  ori $5, $0, 156 | @00003000: $ 1 <= 00000004  @00003004: $31 <= 0000300c  @00003010: \*00000008 <= 0000300c  @00003014: $ 5 <= 0000009c |
| Tuse\_rt=2，Tnew\_W=ALU，转发 | |
| ori $1, $0, 4  ori $5, $0, 256  addu $3, $1, $1 ##  ori $4, $0, 28  addu $7, $1, $5  sw $3, 4($5) ## | @00003000: $ 1 <= 00000004  @00003004: $ 5 <= 00000100  @00003008: $ 3 <= 00000008  @0000300c: $ 4 <= 0000001c  @00003010: $ 7 <= 00000104  @00003014: \*00000104 <= 00000008 |
| Tuse\_rt=2，Tnew\_W=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1243  sw $2, -4($1)  lw $3, -4($1) ##  ori $9, $0, 156  addu $5, $1, $2  sw $3, 8($1) ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001243  @00003008: \*00000000 <= 00001243  @0000300c: $ 3 <= 00001243  @00003010: $ 9 <= 0000009c  @00003014: $ 5 <= 00001247  @00003018: \*0000000c <= 00001243 |
| Tuse\_rt=2，Tnew\_W=PC，转发 | |
| ori $1, $0, 4  jal label ##  nop  addu $3, $1, $1  label:  addu $8, $1, $1  sw $31, 4($1) ##  ori $5, $0, 156 | @00003000: $ 1 <= 00000004  @00003004: $31 <= 0000300c  @00003010: $ 8 <= 00000008  @00003014: \*00000008 <= 0000300c  @00003018: $ 5 <= 0000009c |

## 思考题

1. 流水线冒险
2. **在采用本节所述的控制冒险处理方式下，PC的值应当如何被更新？请从数据通路和控制信号两方面进行说明。**

答：

从数据通路方面，由于寄存器值比较提前至位于D级，考虑将NPC部件同样置于D级，这样便于保证比较与跳转操作的接续性，此处注意beq判断为0时npc应输出pc4+4，jal等写入寄存器的指令地址也应该是pc4+4。由于延迟槽的使用，在未产生跳转需求时应保持pc在每个周期+4的操作，因此我在F级设置了add4部件，将pc输出值+4后再输入到pc中。当产生跳转需求时，PC的输入端口面临pc4和npc两个选择，因此我设置了MUX\_PcSel多路选择器用于选择次指令地址。由于需要暂停控制，还需要从暂停控制器接到PC的通路，接入使能信号用于控制暂停。

从控制信号方面，将D级指令信号译码得到NPC操作的控制信号，目前分为三种（`bType\_npc,`jType\_npc,`rType\_npc），在MUX\_PcSel多路选择器中需要译码得到的选择信号，`pc4\_pc表示使用pc+4（此处pc是F级的pc），`npc表示使用跳转指令地址（此处pc是D级的pc）。PC中的使能信号为暂停信号取反，当使能信号为1时PC正常工作，为0时PC值冻结。

1. **对于jal等需要将指令地址写入寄存器的指令，为什么需要回写PC+8？**

答：

由于使用了延迟槽，CPU会自动执行跳转指令后一个指令，该指令是在编译过程中编译器优化加入的，因此返回至跳转位置时应执行的是跳转指令后第二条指令，即pc+8，所以回写入寄存器的指令应是pc+8。

1. 数据冒险的分析
2. **为什么所有的供给者都是存储了上一级传来的各种数据的**流水级寄存器**，而不是由ALU或者DM等部件来提供数据？**

答：

因为可能产生死循环：若供给者为ALU，需求者也为ALU，此时ALU的输出端口与输入端口相接产生错误。而所有供给者都设置为流水级寄存器在我看来是为了功能上的协调统一，否则相同的效率下暂停与转发的控制将更加复杂难以操作。

1. AT法处理流水线数据冒险
2. **如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。**

答：

若上一级的数据来自W级转发而之后并未使用转发过的数据，则可能会导致数据的错误。比如序列：

ori $1, $0, 0x1234

addu $1, $2, $3 （$2+$3=0x5678）

xxx

xxx

addu $5, $1, $2

当第二条指令位于W级时，第五条指令位于D级，此时原始数据RD1为0x1234，转发后数据V1为0x5678，下一个时钟周期，若使用原始数据，第五条指令到达E级，此时参与计算的$1为0x1234，但实际上第二条指令刚刚写入了寄存器，$1值实际为0x5678，因此产生错误。

1. **我们为什么要对GPR采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？**

答：

若某一时钟周期，GRF需要写入寄存器，同时有新的指令需要在D级读取该寄存器的值，时钟上升沿到来时，两操作并行导致读取的寄存器值仍为原来的值，可能产生错误。根据理论课知识，如果不采用内部转发，似乎需要使用反时钟，使得用于写入的正时钟出于下降沿时，用于读取的反时钟出于上升沿，此时再读取可得到已写入的寄存器值（但似乎这样过于复杂，且时钟沿的延迟不易把控？因此放弃这一方案采用内部转发）。

1. **为什么0号寄存器需要特殊处理？**

答：

因为0号寄存器无法写入，且任何时候读出均为0，不涉及数据冲突与冒险，若未进行特殊处理，当某一指令需要写入0寄存器，且后面指令需要读取0号寄存器时，根据普通转发机制，产生转发，将要写入的值转发，导致参与运算或比较的数据不是应读出的0，可能产生错误。

1. **什么是“最新产生的数据”？**

答：

我的理解是在目前时序中最后写入寄存器的值，也即在使用数据的部件之后离它越近，产生的数据越新，而流水下来的数据最“旧”。

1. **在AT方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为0”，但在CPU写入GRF的时候，是有一个we信号来控制是否要写入的。为何在AT方法中不需要特判we呢？为了**用且仅用**A和T完成转发，在翻译出A的时候，要结合we做什么操作呢？**

答：

我的理解是供给者既然已经称为了供给者，说明指令在它的位置上产生的值在之后一定会有写入寄存器的操作（若不被转发覆盖），因此不需要写使能的特判。为了仅用AT完成转发，我认为在翻译A时，若WE为1，则照常翻译A，若WE为0，则直接将A翻译成0，即写入0寄存器，在判断转发时也将直接被判定为不转发。但其实我的设计中并没有使用此方法，而是在T的翻译中加入了写使能的考虑，由于Tnew代表产生写入寄存器值的部件，我将ALU、PC、DM三个部件信号宏定义为非0值，对于不写入寄存器的指令，其Tnew记为0，在选择转发内容时，0即代表不转发，从而完成转发的控制。

1. 在线测试相关说明
2. **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？如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略，**比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。此思考题请同学们结合自己测试CPU使用的具体手段，按照自己的实际情况进行回答。**

答：

在实验中，我的测试是根据策略矩阵进行构造的，在文档的前面列出了每种可能的一个典型测试样例。针对每个Tuse和Tnew，有固定数目的指令组合，我根据这些指令组合构造了大量测试样例，但实际上，每个测试样例只能测试一种跳转或暂停操作，由于水平欠缺，不会进行自动化测试，无法保证测试的操作覆盖每个寄存器，这是本实验的不足之处。此外，考虑到之后需要添加大量指令，手动构造测试样例工作将变得异常繁重，或将带来巨大隐患。