# 计算机组成原理P6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{LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、 SUBU、 MULT、 MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}。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、MD部件（用于计算乘除法）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 |
| instr[31:0] | O | 32位输入当前指令，用于判断选择比较操作 |
| zero | O | 1位输出判断结果，为0表示不跳转，1表示跳转 |

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. MD

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| instr[31:0] | I | 32位输入当前指令，译码出操作选择信号 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| D1[31:0] | I | 32位输入第一个操作数 |
| D2[31:0] | I | 32位输入第二个操作数 |
| HI[31:0] | O | 32位输出HI读取结果 |
| LO[31:0] | O | 32位输出LO读取结果 |
| start | O | 1位输出表示乘除运算开始 |
| busy | O | 1位输出表示乘除模块工作状态 |

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（目前全部来自rt寄存器） |
| 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 |
| HI[31:0] | I | 32位输入MD读取HI寄存器结果 |
| LO[31:0] | I | 32位输入MD读取LO寄存器结果 |
| 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 |
| HI[31:0] | I | 32位输出M级HI |
| LO[31:0] | I | 32位输出M级LO |
| 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 |
| HI[31:0] | I | 32位输入M级HI |
| LO[31:0] | I | 32位输入M级LO |
| 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 |
| HI[31:0] | I | 32位输出W级HI |
| LO[31:0] | I | 32位输出W级LO |
| 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 |
| HI[31:0] | I | 32位输入HI |
| LO[31:0] | I | 32位输入LO |
| 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 |
| HI\_M | I | 32位输入来自M级的HI |
| LO\_M | I | 32位输入来自M级的LO |
| 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 |
| HI\_W | I | 32位输入来自W级的HI |
| LO\_W | I | 32位输入来自W级的LO |
| 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 |
| HI\_M | I | 32位输入来自M级的HI |
| LO\_M | I | 32位输入来自M级的LO |
| 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 |
| HI\_W | I | 32位输入来自W级的HI |
| LO\_W | I | 32位输入来自W级的LO |
| 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 |
| HI\_M | I | 32位输入来自M级的HI |
| LO\_M | I | 32位输入来自M级的LO |
| 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 |
| HI\_W | I | 32位输入来自W级的HI |
| LO\_W | I | 32位输入来自W级的LO |
| 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 |
| HI\_M | I | 32位输入来自M级的HI |
| LO\_M | I | 32位输入来自M级的LO |
| 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 |
| HI\_W | I | 32位输入来自W级的HI |
| LO\_W | I | 32位输入来自W级的LO |
| 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 |
| HI\_W | I | 32位输入来自W级的HI |
| LO\_W | I | 32位输入来自W级的LO |
| 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

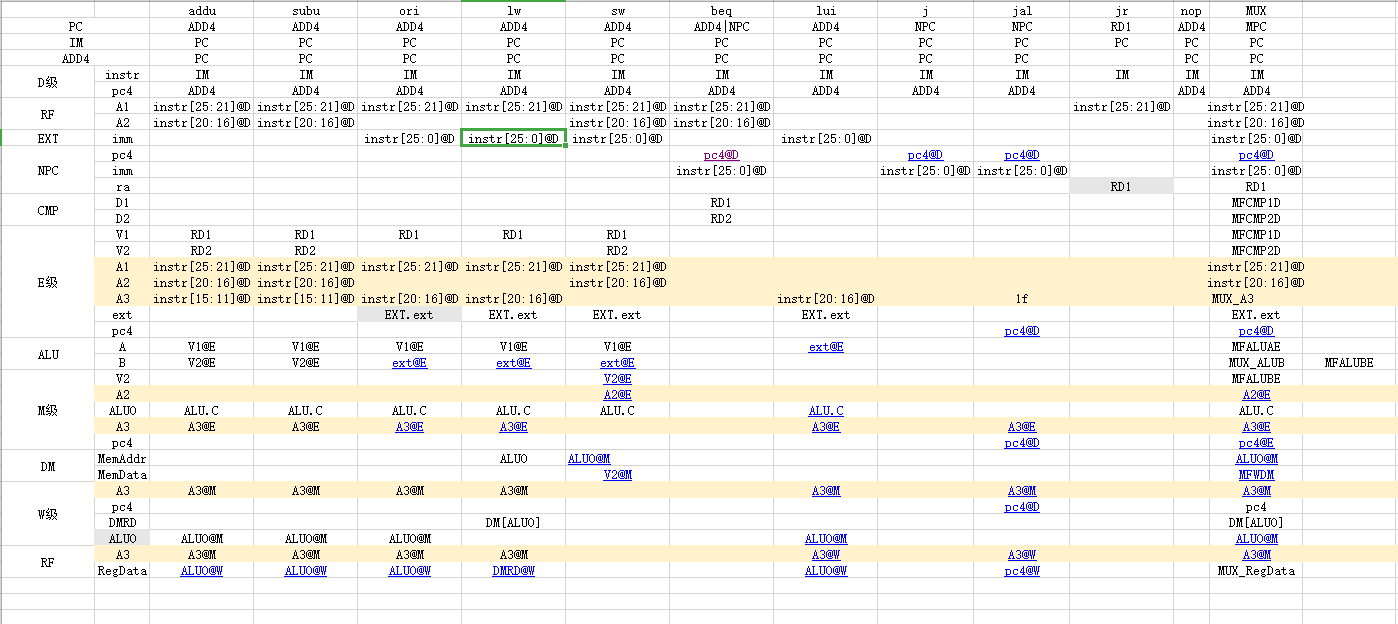
|  |  |  |
| --- | --- | --- |
| 功能描述 | 放眼全局，用于暂停控制信号的生成 | |
| 信号名 | 方向 | 描述 |
| instr\_D[31:0] | I | 32位输入位于D级的指令 |
| instr\_E[31:0] | I | 32位输入位于E级的指令 |
| instr\_M[31:0] | I | 32位输入位于M级的指令 |
| A3\_E[4:0] | I | 5位输入E级指令的写入寄存器地址 |
| A3\_M[4:0] | I | 5位输入M级指令的写入寄存器地址 |
| state\_md | I | 1位输入表示乘法模块工作状态的信号 |
| 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 |

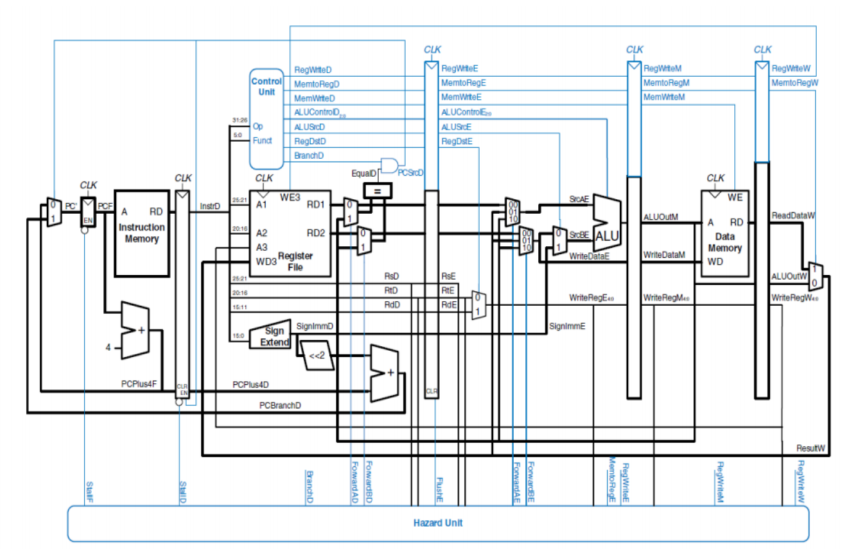
1. DECODE

|  |  |  |
| --- | --- | --- |
| 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 |
| DMLOp[3:0] | O | 4位输出DM的load操作选择信号 |
| DMSOp[3:0] | O | 4位输出DM的store操作选择信号 |
| MDOp[3:0] | O | 4位输出MD的操作选择信号 |
| start | O | 1位输出乘除模块工作开始信号 |
| 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 |
| ismd | O | 1位输出乘除法指令判断信号 |

### 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

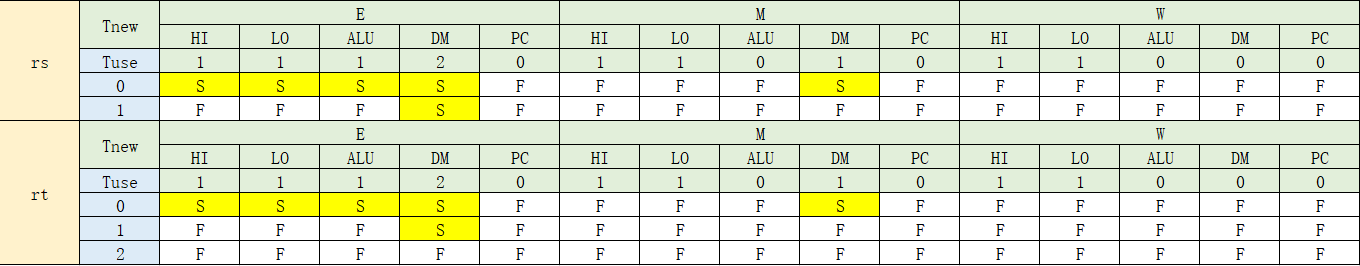
1. 跳转

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。其中，对于b类指令，根据CMP模块的判断结果配合进行NPC模块的次指令地址选择。

1. 暂停

（部分指令的Tuse和Tnew表）





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。

本实验中添加了乘除模块，根据现实情况，乘除模块工作时间较长（乘法为5个时钟周期，除法为10个时钟周期，此外还有开始工作的一个时钟周期），当乘除模块工作时，若新指令也需要使用乘除模块，则产生冲突，需要暂停，因此将乘除模块的busy、start信号连入暂停控制部件，当二者之并为1且D级指令为乘除指令时暂停。

此处需注意，当产生冲突的寄存器为0号寄存器时，不暂停。

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。

转发的“供给者”包括M级ALU、PC4、HI、LO，W级ALU、PC4、DM、HI、LO，“需求者”包括D级的cmp1，cmp2，E级的alua、alub，M级的dm。

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 |

乘除模块测试：

|  |  |
| --- | --- |
| ori $2, $0, 6  ori $3, $0, 2  ori $4, $0, 3  ori $31, $0, 1024  mult $3, $4  mflo $5 ##  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $11, $0, 0xc0f  ori $15, $0, 4  mult $11, $15  mflo $12 ##  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326  ori $13, $0, 4  mfhi $13  sw $2, 4($13)  mthi $2  mtlo $3  div $2, $3  mfhi $22  mflo $23  ori $14, $0, 256  lui $15, 0xffff  mult $15, $14  mfhi $26  mflo $27  multu $15, $14  mthi $26  mflo $27  divu $15, $14  mthi $9  mfhi $26  mflo $27  div $15, $14  mtlo $31  mfhi $26  mflo $27  sw $31, -4($31)  div $15, $14  mtlo $31  mfhi $26  mflo $27  lw $30, -4($31) | @00003000: $ 2 <= 00000006  @00003004: $ 3 <= 00000002  @00003008: $ 4 <= 00000003  @0000300c: $31 <= 00000400  @00003014: $ 5 <= 00000006  @0000301c: $10 <= 00000005  @00003024: $11 <= 00000c0f  @00003028: $15 <= 00000004  @00003030: $12 <= 0000303c  @00003038: $ 1 <= 00000009  @0000303c: $ 9 <= 22220000  @00003040: $12 <= 00000146  @00003044: $13 <= 00000004  @00003048: $13 <= 00000000  @0000304c: \*00000004 <= 00000006  @0000305c: $22 <= 00000000  @00003060: $23 <= 00000003  @00003064: $14 <= 00000100  @00003068: $15 <= ffff0000  @00003070: $26 <= ffffffff  @00003074: $27 <= ff000000  @00003080: $27 <= ff000000  @0000308c: $26 <= 22220000  @00003090: $27 <= 00ffff00  @0000309c: $26 <= 00000000  @000030a0: $27 <= 00000400  @000030a4: \*000003fc <= 00000400  @000030b0: $26 <= 00000000  @000030b4: $27 <= 00000400  @000030b8: $30 <= 00000400 |

DM新指令测试：

|  |  |
| --- | --- |
| ori $30, $0, 256  ori $1, $0, 0x1234  lui $2, 0x5678  addu $3, $1, $2  ori $4, $0, 0x98a1  lui $5, 0xb2c3  addu $6, $4, $5  ori $15, $0, 4  sw $3, 4($15)  sw $6, 8($15)  lw $21, 4($15)  addu $31, $21, $30  lbu $21, 4($15)  addu $31, $21, $30  lbu $21, 5($15)  addu $31, $21, $30  lbu $21, 6($15)  addu $31, $21, $30  lbu $21, 8($15)  addu $31, $21, $30  lbu $21, 9($15)  addu $31, $21, $30  lbu $21, 6($15)  addu $31, $21, $30  lbu $21, 8($15)  addu $31, $21, $30  lbu $21, 10($15)  addu $31, $21, $30  sb $6, 0($15)  sb $6, 1($15)  sb $6, 2($15)  sb $6, 3($15)  sh $6, 12($15)  sh $6, 14($15)  lw $25, 0($15)  lw $25, 4($15)  lw $25, 8($15)  lw $25, 12($15) | @00003000: $30 <= 00000100  @00003004: $ 1 <= 00001234  @00003008: $ 2 <= 56780000  @0000300c: $ 3 <= 56781234  @00003010: $ 4 <= 000098a1  @00003014: $ 5 <= b2c30000  @00003018: $ 6 <= b2c398a1  @0000301c: $15 <= 00000004  @00003020: \*00000008 <= 56781234  @00003024: \*0000000c <= b2c398a1  @00003028: $21 <= 56781234  @0000302c: $31 <= 56781334  @00003030: $21 <= 00000034  @00003034: $31 <= 00000134  @00003038: $21 <= 00000012  @0000303c: $31 <= 00000112  @00003040: $21 <= 00000078  @00003044: $31 <= 00000178  @00003048: $21 <= 000000a1  @0000304c: $31 <= 000001a1  @00003050: $21 <= 00000098  @00003054: $31 <= 00000198  @00003058: $21 <= 00000078  @0000305c: $31 <= 00000178  @00003060: $21 <= 000000a1  @00003064: $31 <= 000001a1  @00003068: $21 <= 000000c3  @0000306c: $31 <= 000001c3  @00003070: \*00000004 <= 000000a1  @00003074: \*00000004 <= 0000a1a1  @00003078: \*00000004 <= 00a1a1a1  @0000307c: \*00000004 <= a1a1a1a1  @00003080: \*00000010 <= 000098a1  @00003084: \*00000010 <= 98a198a1  @00003088: $25 <= a1a1a1a1  @0000308c: $25 <= 56781234  @00003090: $25 <= b2c398a1  @00003094: $25 <= 98a198a1 |

## 思考题

（一）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？

答：

一方面，根据MIPS指令集，ALU的工作模式是直接计算并立即将计算结果传出使写入某一寄存器，而乘除法mult、multu、div、divu并非在计算结束后直接将结果写入寄存器，而是需要通过新指令mflo、mfhi将最近产生的结果写入寄存器，这与ALU的工作步骤不符，因此无法直接整合进ALU。另一方面，在实际工程中，乘除法并不是verilog中乘除符号一样的简单计算，其计算过程较长（本实验中为5周期或10周期），而ALU中其他计算均为1周期结束，因此二者的时序不符，无法将乘除法部件加入到ALU中。

Verilog建模的流水线CPU与MIPS的CPU构成基本相符，HI、LO两个独立寄存器也正是对MIPS的设置的遵从。对于乘法，两32位数相乘其结果位数最高可能达到64位，因此计算需要两个32位变量分别存储高32位和低32位；对于除法，将产生商和余数两个32位数，同样需要两个32位变量分别存储。与此同时，根据MIPS指令的规则，一条指令最多只能存入一个寄存器，因此在乘除模块中就需要两个独立的寄存器存储计算产生的两个结果，之后再根据需要决定是否、什么时候将这两个独立寄存器中的数据存入到GRF中，这样也更加快速，无需再次计算。

（二）参照你对延迟槽的理解，试解释“乘除槽”。

答：

乘除法在乘除模块中开始进行后，需要5或10周期才能产生结果，在这个时间中，后面的指令可以继续进入流水线，此时需要在D级对指令进行判断，如果为乘除指令，则因乘除模块的占用而暂停，若为其他指令，其操作与乘除不相干扰，可以继续执行，由此可以提升CPU的效率。

（三）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）

答：

由c语言中的字符串可知，一个英文字母占用一个字节，因此在处理字符串时需要以字节为单位进行处理，尤其是当处理数量不为四的倍数时按字节访问在性能上更具优势。

（四）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？

答：

延用P5的思考方法，将指令分为8类：

|  |  |
| --- | --- |
| cal\_r | add、addu、andd、norr、orr、slt、sltu、sllv、srav、srlv、sub、subu、xorr |
| cal\_i | addi、addiu、andi、lui、ori、slti、sltiu、xori |
| load | lb、lbu、lh、lhu、lw |
| store | sb、sh、sw |
| bType | beq、bgez、bgtz、blez、bltz、bne |
| jType | jr、jalr、j、jal |
| shift | sll、sra、srl |
| Ismd（乘除模块） | div、divu、mfhi、mflo、mthi、mtlo、mult、multu |

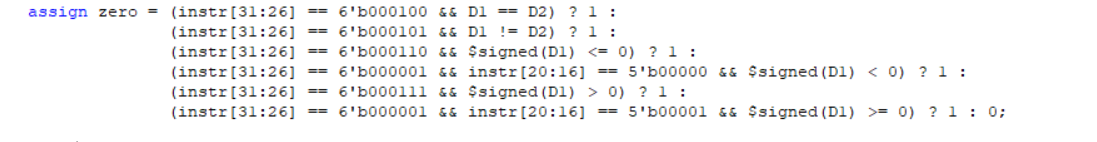
根据策略矩阵与指令分类构造测试样例，重点测试新加入的乘除模块。

1. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

答：

首先将数十条代码分为了八类（详见上一个问题），相同类中指令的数据通路基本相同，以类名变量代替这十几个变量，进而加入到控制信号的表达式中，因此在添加指令时只需要将新指令添加到其所属类的表达式中即可，有效减少工作量。

对于b类指令，不同跳转的判断内容不同，我在CMP模块中进行了译码，直接将六个b类指令翻译出来，针对不同指令进行相应的判断，判断结果全部用一个变量zero表示：



这样也可以避免NPC模块中控制信号的增加导致的混乱。

对于有共性的内容，在一个模块中可以先将共同的内容表示出来，之后根据不同的操作信号进行进一步的处理，比如DM模块中各指令都需要数据存储器中相应整地址读取的数据，因此可以将其先赋给变量readword，之后根据不同指令要求进行进一步的截取或拼接。

将指令分类，各类别的AT基本完全相同，因此可以构造简单的策略矩阵，这也使得本CPU的冲突部分代码量很少，无需逐条考虑冲突，而且在构造测试样例时也可以根据指令类别构造出一整类的覆盖性测试样例，减轻思考负担。

各类别指令的数据通路基本相同，因此在译码时也可直接将一整类添加至操作信号表达式中，但与此同时，可以根据类别内部指令的差异性构造操作信号，我认为将每个指令单独配置操作信号可以有效降低思考量与错误率（或许增加一些代码量也是降低复杂性的一种方法）。