# 计算机组成原理P7实验报告

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为Verilog实现的包含中断与异常处理的流水线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、ERET、MFC0、MTC0}。

在顶层模块mips.v下分CPU、BRIDGE、Timer0、Timer1四个部分，其中Timer0与Timer1负责提供外部中断，由桥与CPU沟通。

为满足中断与异常的处理，CPU中增加了关键模块CP0，置于数据通路的M级，内设SR、CAUSE、EPC、PRId四个寄存器。PC模块增添判断pc引发的取指异常的功能，D级增设ckeckD模块用于判断未知指令码的异常，E级增设checkE模块，用于判断ALU加减法运算异常，M级增设checkM模块，用于判断DM存取异常。

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 |

1. CP0

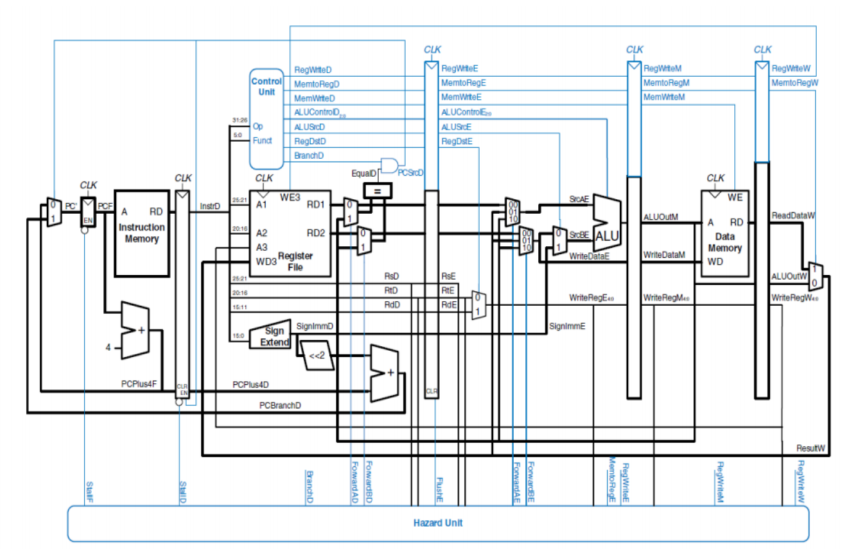
|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于存储或读取异常、中断产生的信息 | |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| instr\_M[31:0] | I | 32位输入M级指令 |
| instr\_W[31:0] | I | 32位输入W级指令 |
| CP0Data[31:0] | I | 32位输入存入CP0的值 |
| HWInt[7:2] | I | 6位输入外部中断信号 |
| exccode[6:2] | I | 5位输入异常种类编码 |
| backPc\_D[31:0] | I | 32位输入D级指令对应的返回pc |
| backPc\_E[31:0] | I | 32位输入E级指令对应的返回pc |
| backPc\_M[31:0] | I | 32位输入M级指令对应的返回pc |
| afterJump\_D | I | 1位输入判断D级指令是否为延迟槽指令 |
| afterJump\_E | I | 1位输入判断E级指令是否为延迟槽指令 |
| afterJump\_M | I | 1位输入判断M级指令是否为延迟槽指令 |
| exccode\_Dn[6:2] | I | 5位输入D级指令对应的异常编码 |
| exccode\_En[6:2] | I | 5位输入E级指令对应的异常编码 |
| exccode\_Mn[6:2] | I | 5位输入M级指令对应的异常编码 |
| macroPc[31:0] | O | 32位输出宏观pc |
| CP0RD[31:0] | O | 32位输出CP0中寄存器读取的值 |
| epc[31:2] | O | 30位输出epc的高30位 |
| exc\_int | O | 1位输出异常和中断总信号 |

1. BRIDGE

|  |  |  |
| --- | --- | --- |
| 功能描述 | 用于连接CPU与外部设备 | |
| 信号名 | 方向 | 描述 |
| interrupt | I | 1位输入外部中断信号 |
| PrAddr[31:2] | I | 30位输入外部设备地址，按字访问 |
| PrWD[31:0] | I | 32位输入存入外部设备的数据 |
| PrWrite | I | 1位输入外部设备写使能 |
| timer0\_RD[31:0] | I | 32位输入Timer0读出数据 |
| Timer1\_RD[31:0] | I | 32位输入Timer1读出数据 |
| timer0\_IRQ | I | 1位输入Timer0中断请求 |
| Timer1\_IRQ | I | 1位输入Timer1中断请求 |
| PrRD[31:0] | O | 32位输出外部设备读取数据 |
| HWInt[7:2] | O | 6位输出总外部中断记录 |
| timer0Write | O | 1位输出Timer0写使能 |
| Timer1Write | O | 1位输出Timer1写使能 |
| device\_Addr[31:2] | O | 30位输出外部设备写入地址 |
| device\_WD[31:0] | O | 32位输出外部设备写入数据 |

### 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

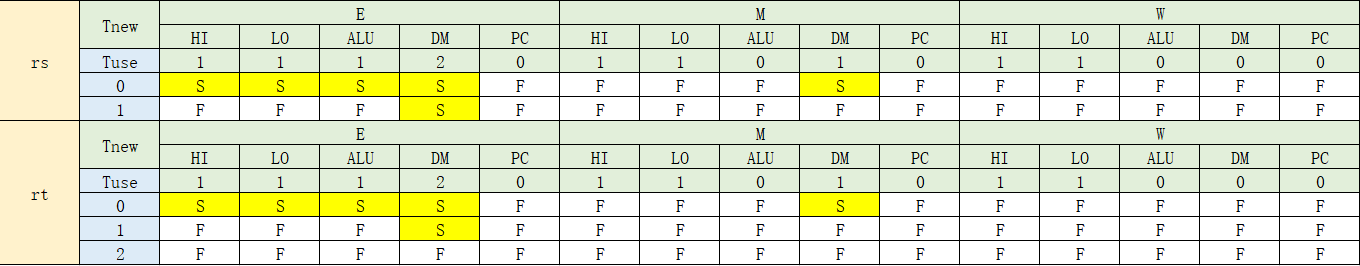
1. 跳转

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

|  |  |
| --- | --- |
| Mips代码 | Verilog运行结果 |
| .text  ori $24, $0, 0x1401  mtc0 $24, $12  ori $1, $0, 11  ori $2, $0, 7  ori $3,$0, 0x7f00  ori $4, $0, 0x7f04  ori $5, $0, 0x7f10  ori $6, $0, 0x7f14  ori $7, $0, 6  sw $1, 0($3)  sw $2, 0($4)  sw $1, 0($5)  sw $7, 0($6)  sw $7, 4($6)  lui $10, 0x7fff  lui $11, 0x7fff  jal label  addu $12, $10, $11  addu $13, $12, $12  label:  ori $15, $0, 0x5000  ori $16, $0, 15  jr $15  nop  sh $16, 0($4)  ori $18, $0, 5  lui $17, 0x8000  jal note  subu $19, $17, $18  ori $20, $0, 15  note:  nop  nop  nop  nop  nop  nop  nop  ori $20, $0, 0x3002  jr $20  nop  addu $10, $11, $12##change to ffffffff  ori $18, $0, 145  ori $27, $0, 265  mfc0, $29, $14  end:  beq $0, $0,end  nop  .ktext 0x4180  mfc0 $31, $14  ori $30, $0, 0x3034  beq $31, $30, special1  ori $30, $0, 0x3040  beq $31, $30, special2  nop  ori $30, $0, 0x3048  beq $31, $30, special3  nop  ori $30, $0, 0x5000  beq $30, $31, special4  nop  ori $30, $0, 0x305c  beq $31, $30, special5  nop  ori $30, $0, 0x3068  beq $31, $30, special6  nop  ori $30, $0, 0x3000  beq $31, $30, special7  nop  ori $30, $0, 0x309c  beq $31, $30 special8  nop  eret  special1:  ori $6, $0, 0x7f00  eret  special2:  ori $11, $0, 0  eret  special3:  ori $12, $0, 20  eret  special4:  ori $15, $0, 0x305c  mtc0 $15, $14  eret  special5:  ori $4, $0, 6  eret  special6:  ori $17, $0, 5  eret    special7:  ori $20, $0, 0x309c  mtc0 $20, $14  eret    special8:  ori $31, $0, 0x30a4  mtc0 $31, $14  eret | @00003000: $24 <= 00001401  @00003008: $ 1 <= 0000000b  @0000300c: $ 2 <= 00000007  @00003010: $ 3 <= 00007f00  @00003014: $ 4 <= 00007f04  @00003018: $ 5 <= 00007f10  @0000301c: $ 6 <= 00007f14  @00003020: $ 7 <= 00000006  @00004180: $31 <= 00003034  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @000041e4: $ 6 <= 00007f00  @00003038: $10 <= 7fff0000  @0000303c: $11 <= 7fff0000  @00003040: $31 <= 00003048  @00003044: $12 <= fffe0000  @0000304c: $15 <= 00005000  @00004180: $31 <= 00003050  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041b0: $30 <= 0000305c  @000041bc: $30 <= 00003068  @000041c8: $30 <= 00003000  @000041d4: $30 <= 0000309c  @00003050: $16 <= 0000000f  @00004180: $31 <= 00005000  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041fc: $15 <= 0000305c  @00004180: $31 <= 0000305c  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041b0: $30 <= 0000305c  @00004208: $ 4 <= 00000006  @0000305c: \*00000004 <= 000f0000  @00003060: $18 <= 00000005  @00003064: $17 <= 80000000  @00003068: $31 <= 00003070  @0000306c: $19 <= 7ffffffb  @00004180: $31 <= 00003074  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041b0: $30 <= 0000305c  @000041bc: $30 <= 00003068  @000041c8: $30 <= 00003000  @000041d4: $30 <= 0000309c  @00004180: $31 <= 00003090  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041b0: $30 <= 0000305c  @000041bc: $30 <= 00003068  @000041c8: $30 <= 00003000  @000041d4: $30 <= 0000309c  @00003090: $20 <= 00003002  @00004180: $31 <= 00003000  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041b0: $30 <= 0000305c  @000041bc: $30 <= 00003068  @000041c8: $30 <= 00003000  @00004218: $20 <= 0000309c  @00004180: $31 <= 0000309c  @00004184: $30 <= 00003034  @0000418c: $30 <= 00003040  @00004198: $30 <= 00003048  @000041a4: $30 <= 00005000  @000041b0: $30 <= 0000305c  @000041bc: $30 <= 00003068  @000041c8: $30 <= 00003000  @000041d4: $30 <= 0000309c  @00004224: $31 <= 000030a4  @000030a4: $27 <= 00000109  @000030a8: $29 <= 000030a4 |

## 思考题

1. 我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？（Tips：什么是接口？和我们到现在为止所学的有什么联系？

答：

硬件/软件接口指将二者联系起来的部分，比如本实验中CPU正常运行的用户态和进入异常的内核态之间的连接部分，我认为可以包括将什么认定为异常、中断的这种法则。

百度：接口（硬件类接口）是指同一[计算机](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA/140338" \t "https://baike.baidu.com/item/%E6%8E%A5%E5%8F%A3/_blank)不同功能层之间的[通信](https://baike.baidu.com/item/%E9%80%9A%E4%BF%A1/300982" \t "https://baike.baidu.com/item/%E6%8E%A5%E5%8F%A3/_blank)规则称为接口。接口（软件类接口）是指对协定进行定义的[引用类型](https://baike.baidu.com/item/%E5%BC%95%E7%94%A8%E7%B1%BB%E5%9E%8B/2889743" \t "https://baike.baidu.com/item/%E6%8E%A5%E5%8F%A3/_blank)。其他类型实现接口，以保证它们支持某些操作。接口指定必须由类提供的成员或实现它的其他接口。与类相似，接口可以包含方法、属性、[索引器](https://baike.baidu.com/item/%E7%B4%A2%E5%BC%95%E5%99%A8/2886331" \t "https://baike.baidu.com/item/%E6%8E%A5%E5%8F%A3/_blank)和事件作为成员。

1. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

答：

现代计算机的DM（主存）位于CPU外部，即与其他外接部件并列。我认为这是因为CPU主要负责运算，其需要较小的容量，较大的速度；而DM主要负责存储，其需要较大的空间，将DM置于CPU内部可能拖慢CPU速度，而且使内部接线更加复杂。

1. BE 部件对所有的外设都是必要的吗？

答：

我认为不是。至少在我目前的CPU设计中并没有使用BE信号。我猜想这是因为外部仅有的TImer部件仅支持整字的存取，但另一方面，正如P6中我的CPU没有设置用于截取位数的扩展DM模块，我认为在理想情况下也可以通过译码出具体指令的方式进行按特定位数存取的操作（当然，现实生活中必然不能这么干，太浪费资源了）。

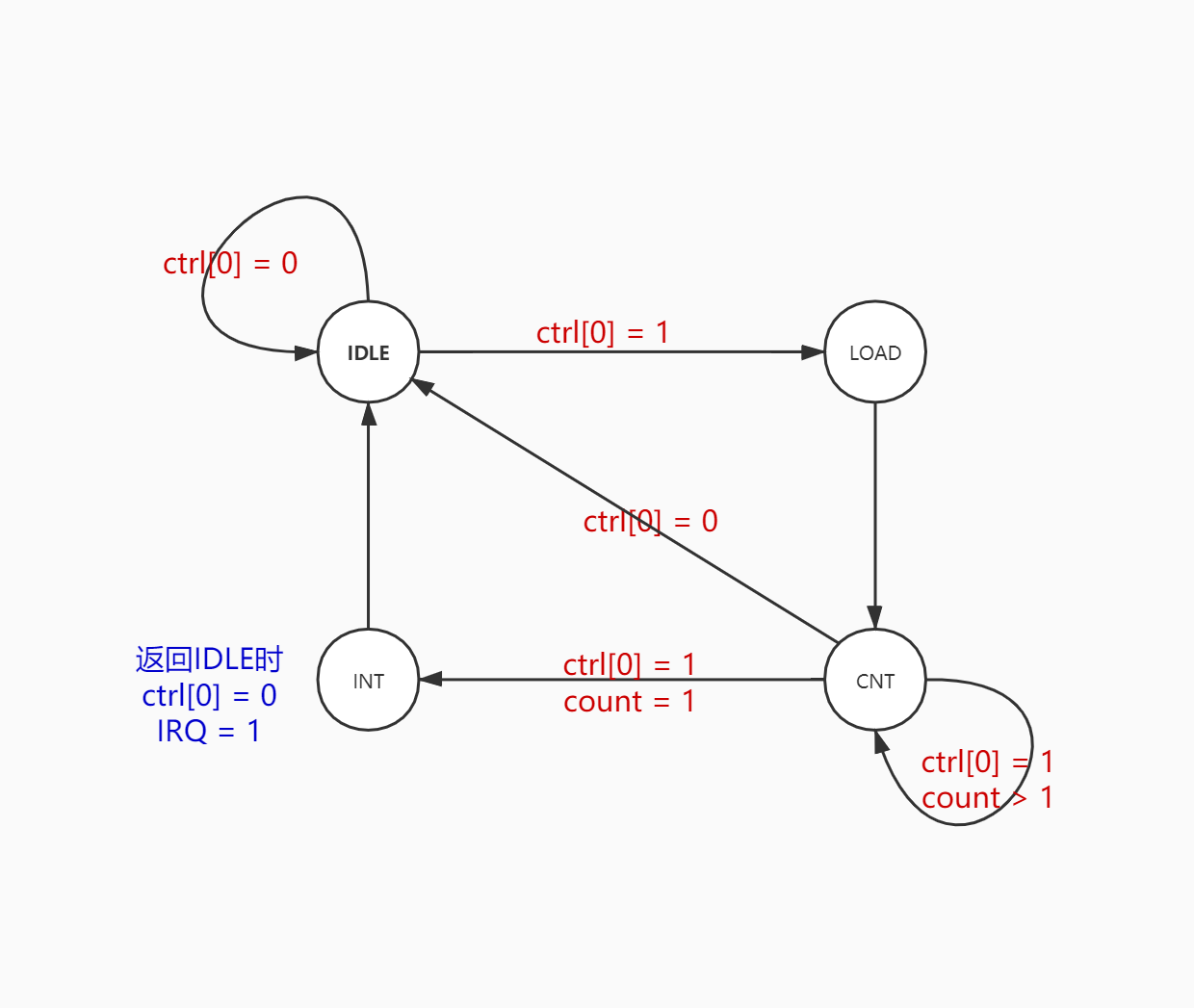
1. 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

答：

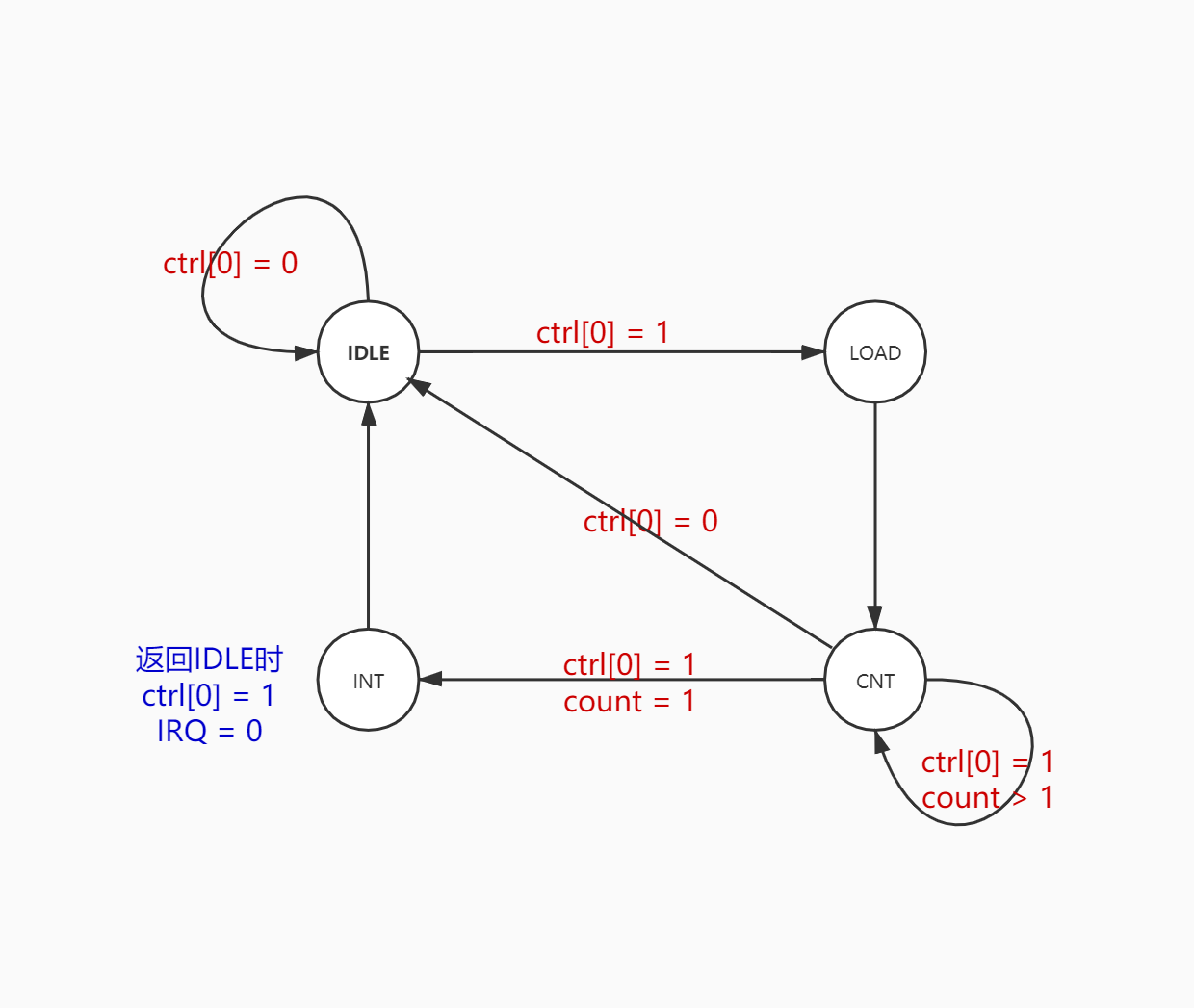
两种中断模式大部分行为是相同的，最初都是处于闲置（IDLE）状态，发出的中断请求保持0，在CONTROL寄存器0位置1后开始工作，即进入加载（LOAD）状态，将预先设定好的计数初始值（preset）存入COUNT寄存器，之后进入计数（CNT）状态，当Timer处于计数状态，且CONTROL的0位（即允许计数）时，count每周期减一，当count减为0时，进入中断（INT）状态，产生中断请求（该中断请求信号与CONTROL的[3]位与运算（考虑中断屏蔽）得到最终由Timer发出的中断信号），在中断状态的下一个周期，状态再次回到闲置状态。

两种中断模式的不同之处就在于进入中断状态后的行为，在INT状态中，如果CONTROL[2:1]=00，即模式0，但CONTROL[0]变为0，即禁止继续计数，状态返回闲置状态，中断请求保持1，若要使Timer重新开始工作，需要再次通过sw改变CONTROL的值；如果CONTROL[2:1]=01，即模式1，中断请求立刻恢复0，状态返回闲置状态，但此时CONTROL[0]为1，Timer将重新开始计数。

模式0状态转移图：



模式1状态转移图：



1. 请开发一个主程序以及定时器的exception handler。

答：

|  |
| --- |
| **主程序** |
| #main  .text  ori $1, $0, 9  ori $2, $0, 0x7f00 #address of Timer0  ori $3, $0, 0x100 #for preset  ori $4, $0, 0xfc02 #for initial SR in CP0  mtc0 $4, $13 #save SR  sw $3, 4($2) #save preset  sw $1, 0($2) #save control  Loop:  beq $0, $0, end  nop  end:  j Loop  nop |
| **exception handler** |
| #exception handler  .ktext 4180  sw $24, 0($sp) #protect now  subi $sp, $sp, 4  sw $25, 0($sp)  subi $sp, $sp, 4  sw $26, 0($sp)  subi $sp, $sp, 4  sw $27, 0($sp)  subi $sp, $sp, 4  sw $28, 0($sp)  subi $sp, $sp, 4  sw $29, 0($sp)  subi $sp, $sp, 4  sw $30, 0($sp)  subi $sp, $sp, 4  mfc0 $26, $13  andi $27, $26, 0x0400 #interrupt at normal instruction  andi $28, $26, 0x80000400 #interrupt at delay instruction  ori $29, $0, 0x400  ori $30, $0, 0x80000400  beq $27, $29, handler  nop  beq $28, $30, handler  nop  handler:  ori $25, $0, 9  ori $24, $0, 0x7f00  sw $25, 0($24) #save control again  addi $sp, $sp, 4 #return before interrupt  lw $30, 0($sp)  addi $sp, $sp, 4  lw $29, 0($sp)  addi $sp, $sp, 4  lw $28, 0($sp)  addi $sp, $sp, 4  lw $27, 0($sp)  addi $sp, $sp, 4  lw $26, 0($sp)  addi $sp, $sp, 4  lw $25, 0($sp)  addi $sp, $sp, 4  lw $24, 0($sp)  eret |

1. 请查阅相关资料，说明鼠标和键盘的输入信号是如何被CPU知晓的？

答：

点击鼠标或敲击键盘与实验中的interrupt信号相似，当鼠标或键盘操作时，产生外部中断信号，此时程序进入内核态，由于鼠标、键盘的操作行为不同，会产生不同的中断信号，CPU通过判断这些信号跳转至不同的内核程序，调用不同的内核代码。