# 计算机组成原理P5流水线CPU实验报告

20373944 何天然

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为Verilog实现的单周期MIPS-CPU，支持的指令集包含{ lw、sw、lb、lbu、sb、lh、lhu、sh、add、addu、sub、subu、and、or、slt、nor、xor、sll、srl、sra、sllv、srlv、srav、sltu、beq、bne、bgtz、blez、bltz、bgez、addi、addiu、andi、ori、xori、lui、slti、sltiu、j、jr、jal、jalr }共42条，其中add和sub不支持溢出中断，其行为与addu和subu完全一致。为了实现这些功能，CPU主要包含 Controller、MainController、HazardSolveUnit、Datapath、IF、IF\_ID、ID、ID\_EX、EX、EX\_MEM、MEM、MEM\_WB、PC、IM、NPC、EXT、CMP、GRF、ALU、DM，这些模块共可分成三层，顶层为Controller和 Datapath；Controller可分为MainController和HazardSolveUnit；Datapath可分为IF、ID、EX、MEM、IF\_ID、ID\_EX、EX\_MEM、MEM\_WB，其中IF包含PC、IM，ID包含NPC、EXT、CMP、GRF，EX包含ALU，MEM包含DM。

### 关键模块定义

### 1. PC（Program Counter）

模块定义：

module PC(

input clk,

input reset,

input WE,

input [31:0] nextPC,

output reg [31:0] PC

);

表1 PC端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | nextPC | I | 32 | 下一条指令的地址 |
| 5 | PC | O | 32 | 当前指令地址 |

内部逻辑说明：

用一个32位寄存器存储当前指令地址，当时钟上升沿到来时，如果同步复位信号有效，则将寄存器复位为起始地址0x00003000，否则若写使能信号有效，将新的指令地址写入寄存器。

### 2. NPC（Next PC）

模块定义：

module NPC(

input [31:0] PC\_F,

input [31:0] PC\_D,

input [31:0] offset,

input [25:0] index,

input [31:0] register,

input [2:0] ctrl,

output [31:0] PCAdd8,

output [31:0] nextPC

);

表2 NPC端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | PC\_F | I | 32 | F级的当前指令地址 |
| 2 | PC\_D | I | 32 | D级的当前指令地址 |
| 3 | offset | I | 32 | branch跳转的偏移量 |
| 4 | index | I | 26 | j或jal跳转地址的2-27位 |
| 5 | register | I | 32 | jr或jalr指令的跳转地址 |
| 6 | ctrl | I | 3 | 选择下条指令的地址 |
| 7 | PCAdd8 | O | 32 | 当前指令地址加8 |
| 8 | nextPC | O | 32 | 下一条指令地址 |

内部逻辑说明：

根据PC\_F或PC\_D分别计算出PC\_F+4、branch跳转地址、j或jal跳转地址、jr或jalr跳转地址，然后根据Ctrl信号选择要输出的下条指令地址nextPC。PCAdd8用于jal和jalr指令将PC\_D+8的值存入相应寄存器。

### 3. IM（Instruction Memory）

模块定义：

module IM(

input [31:0] A,

output [31:0] RD

);

表3 IM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | A | I | 32 | 32位指令地址 |
| 2 | RD | O | 32 | 32位指令 |

内部逻辑说明：

IM存储所有指令。根据32位地址的[13:2]位取出相应指令。

### 4. GRF（General Register File）

模块定义：

module GRF(

input clk,

input reset,

input WE,

input [4:0] A1,

input [4:0] A2,

input [4:0] A3,

input [31:0] WD,

input [31:0] PC,

output [31:0] RD1,

output [31:0] RD2

);

表4 GRF端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | A1 | I | 5 | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1 |
| 5 | A2 | I | 5 | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2 |
| 6 | A3 | I | 5 | 5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器 |
| 7 | WD | I | 32 | 32位数据输入信号 |
| 8 | PC | I | 32 | 当前指令地址 |
| 9 | RD1 | O | 32 | 输出A1指定的寄存器中的32位数据 |
| 10 | RD2 | O | 32 | 输出A2指定的寄存器中的32位数据 |

内部逻辑说明：

通过多路分解器进行输入数据WD和使能信号WE的分配，A3为选择信号；通过多路选择器进行输出数据RD1和RD2的选择，A1和A2分别为选择信号。当时钟上升沿到来时，如果同步复位信号有效，则将所有寄存器的值清零，否则将WD写入A3对应的寄存器。PC信号仅用于格式化输出。

注：使能信号恒为1，是否要进行写入通过地址是否为0来判断。实现了内部转发，当写入地址与读取地址相同时，直接输出WD。

### 5. ALU（Arithmetic Logical Unit）

模块定义：

module ALU(

input [31:0] srcA,

input [31:0] srcB,

input [3:0] ctrl,

output [31:0] result

);

表5 ALU端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | srcA | I | 32 | 操作数A |
| 2 | srcB | I | 32 | 操作数B |
| 3 | ctrl | I | 4 | 控制信号 |
| 4 | result | O | 32 | 计算结果 |

内部逻辑说明：

支持加、减、与、或、异或、移位、比较等运算。分开进行每种计算，最后通过多路选择器和ctrl信号来选择输出结果。

### 6. DM（Data Memory）

模块定义：

module DM(

input clk,

input reset,

input WE,

input [31:0] A,

input [1:0] byteSel,

input [2:0] ctrl,

input [31:0] WD,

input [31:0] PC,

output [31:0] RD

);

表6 DM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | A | I | 32 | 32位地址输入信号 |
| 5 | ctrl | I | 3 | 控制信号 |
| 6 | WD | I | 32 | 32位数据输入信号 |
| 7 | PC | I | 32 | 当前指令地址 |
| 8 | RD | O | 32 | 32位数据输出信号 |

内部逻辑说明：

用一常量ADDRBITS记录地址位数，根据地址位数确定内存的大小，并从32位地址信号A中取出 [ADDRBITS-1:2] 位作为字地址，A的[1:0]位用于字节选择。当时钟上升沿到来时，如果同步复位信号有效，则清空内存，否则若写使能信号有效，则进行数据写入。根据控制信号可对字、半字、字节进行读写。

### 7. EXT（Extender）

模块定义：

module EXT(

input [15:0] imm16,

input [2:0] ctrl,

output [31:0] imm32

);

表7 EXT端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | imm16 | I | 16 | 16位立即数输入信号 |
| 2 | ctrl | I | 3 | 控制信号 |
| 3 | imm32 | O | 32 | 32位立即数输出信号 |

内部逻辑说明：

根据控制信号，可将16位立即数零扩展、符号扩展、加载到高位、左移两位并进行符号扩展，最后输出32位立即数。

### CMP（Comparer）

模块定义：

module ALU(

input [31:0] srcA,

input [31:0] srcB,

input [3:0] ctrl,

output result

);

表8 CMP端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | srcA | I | 32 | 操作数A |
| 2 | srcB | I | 32 | 操作数B |
| 3 | ctrl | I | 4 | 控制信号 |
| 4 | result | O | 1 | 比较结果 |

内部逻辑说明：

支持各种比较运算，如无符号比较、有符号比较、与零比较等。分开进行每种比较，最后通过多路选择器和ctrl信号来选择输出结果。

### IF（Instruction Fetch）

模块定义：

module IF (

input clk,

input reset,

input PCWrite,

input [31:0] nextPC,

output [31:0] PC,

output [31:0] instr

);

表9 IF端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | PCWrite | I | 1 | PC的写使能信号 |
| 4 | nextPC | I | 32 | 下一条指令地址 |
| 5 | PC | O | 32 | 当前指令地址 |
| 6 | instr | O | 32 | 当前指令 |

内部逻辑说明：

为流水线的F级。内部连接PC、IM等模块，用于取指令。

### ID（Instruction Decoder）

模块定义：

module ID (

input clk,

input reset,

input [31:0] PC\_F,

input [31:0] PC\_D,

input [31:0] PC\_W,

input [31:0] instr,

input [4:0] regAddr,

input [31:0] EXBack,

input [31:0] MEMBack,

input [31:0] WBBack,

input regWrite,

input [1:0] regAddrSel,

input [3:0] CMPCtrl,

input [2:0] EXTCtrl,

input [2:0] NPCCtrl,

input [1:0] regRD1Forward,

input [1:0] regRD2Forward,

output [4:0] shamt,

output [4:0] regA1,

output [4:0] regA2,

output [4:0] regA3,

output [31:0] regRD1,

output [31:0] regRD2,

output [31:0] imm32,

output CMPResult,

output [31:0] PCAdd8,

output [31:0] nextPC

);

表10 ID端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | PC\_F | I | 32 | F级的PC |
| 4 | PC\_D | I | 32 | D级的PC |
| 5 | PC\_W | I | 32 | W级的PC |
| 6 | instr | I | 32 | 指令 |
| 7 | regAddr | I | 5 | GRF写入地址 |
| 8 | EXBack | I | 32 | 从E级转发的数据 |
| 9 | MEMBack | I | 32 | 从M级转发的数据 |
| 10 | WBBack | I | 32 | 从W级转发的数据 |
| 11 | regWrite | I | 1 | GRF写使能信号 |
| 12 | regAddrSel | I | 2 | GRF写入地址选择 |
| 13 | CMPCtrl | I | 4 | CMP控制信号 |
| 14 | EXTCtrl | I | 3 | EXT控制信号 |
| 15 | NPCCtrl | I | 3 | NPC控制信号 |
| 16 | regRD1Forward | I | 2 | GRF[rs]转发数据选择信号 |
| 17 | regRD2Forward | I | 2 | GRF[rt]转发数据选择信号 |
| 18 | shamt | O | 5 | 移位位数 |
| 19 | regA1 | O | 5 | rs |
| 20 | regA2 | O | 5 | rt |
| 21 | regA3 | O | 5 | 写入地址 |
| 22 | regRD1 | O | 32 | 转发后的GRF[rs] |
| 23 | regRD2 | O | 32 | 转发后的GRF[rt] |
| 24 | Imm32 | O | 32 | 32位立即数 |
| 25 | CMPResult | O | 1 | CMP比较结果 |
| 26 | PCAdd8 | O | 32 | PC+8 |
| 27 | nextPC | O | 32 | 下一条PC |

内部逻辑说明：

为流水线的D级。内部连接CMP、EXT、GRF、NPC等模块，进行指令的解码、W级的写回。

### EX（Execute）

模块定义：

module EX (

input [4:0] shamt,

input [31:0] regRD1\_orig,

input [31:0] regRD2\_orig,

input [31:0] imm32,

input [31:0] MEMBack,

input [31:0] WBBack,

input ALUSrcASel,

input ALUSrcBSel,

input [3:0] ALUCtrl,

input [1:0] regRD1Forward,

input [1:0] regRD2Forward,

output [31:0] ALUResult,

output [31:0] regRD2

);

表11 EX端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | shamt | I | 5 | 移位位数 |
| 2 | regRD1\_orig | I | 32 | 上一级的GRF[rs] |
| 3 | regRD2\_orig | I | 32 | 上一级的GRF[rt] |
| 4 | imm32 | I | 32 | 32位立即数 |
| 5 | MEMBack | I | 32 | 从M级转发的数据 |
| 6 | WBBack | I | 32 | 从W级转发的数据 |
| 7 | ALUSrcASel | I | 1 | ALU操作数A选择信号 |
| 8 | ALUSrcBSel | I | 1 | ALU操作数B选择信号 |
| 9 | ALUCtrl | I | 4 | ALU控制信号 |
| 10 | regRD1Forward | I | 2 | GRF[rs]转发数据选择信号 |
| 11 | regRD2Forward | I | 2 | GRF[rt]转发数据选择信号 |
| 12 | ALUResult | O | 32 | ALU运算结果 |
| 13 | regRD2 | O | 32 | 转发后的GRF[rt] |

内部逻辑说明：

为流水线的E级。内部连接ALU和其他模块，用于选择操作数、进行计算并输出结果。

### MEM（Memory）

模块定义：

module MEM (

input clk,

input reset,

input [31:0] ALUResult,

input [31:0] regRD2\_orig,

input [31:0] PC,

input [31:0] WBBack,

input memWrite,

input [2:0] DMCtrl,

input regRD2Forward,

output [31:0] memRD

);

表12 MEM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | ALUResult | I | 32 | ALU计算结果 |
| 4 | regRD2\_orig | I | 32 | 上一级的GRF[rt] |
| 5 | PC | I | 32 | 当前指令地址 |
| 6 | WBBack | I | 32 | 从W级转发的数据 |
| 7 | memWrite | I | 1 | DM写使能信号 |
| 8 | DMCtrl | I | 3 | DM控制信号 |
| 9 | regRD2Forward | I | 1 | GRF[rt]转发数据选择信号 |
| 10 | memRD | O | 32 | 内存数据输出信号 |

内部逻辑说明：

为流水线的MEM级。内部连接DM和其他模块，用于内存的读写。

### IF\_ID（Pipeline Register IF\_ID）

模块定义：

module IF\_ID(

input clk,

input reset,

input WE,

input [31:0] instr\_I,

input [31:0] PC\_I,

output reg [31:0] instr\_O,

output reg [31:0] PC\_O

);

表13 IF\_ID端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | instr\_I | I | 32 | 指令 |
| 5 | PC\_I | I | 32 | 指令地址 |
| 6 | instr\_O | O | 32 | 指令 |
| 7 | PC\_O | O | 32 | 指令地址 |

内部逻辑说明：

为流水寄存器IF\_ID，在F级和D级间流水数据。

### ID\_EX（Pipeline Register ID\_EX）

模块定义：

module ID\_EX(

input clk,

input reset,

input WE,

input [4:0] shamt\_I,

input [4:0] regA1\_I,

input [4:0] regA2\_I,

input [4:0] regA3\_I,

input [31:0] regRD1\_I,

input [31:0] regRD2\_I,

input [31:0] imm32\_I,

input [31:0] PCAdd8\_I,

input [31:0] PC\_I,

input memWrite\_I,

input [1:0] EXBackSel\_I,

input [1:0] MEMBackSel\_I,

input [1:0] WBBackSel\_I,

input ALUSrcASel\_I,

input ALUSrcBSel\_I,

input [3:0] ALUCtrl\_I,

input [2:0] DMCtrl\_I,

input [2:0] Tnew\_I,

output reg [4:0] shamt\_O,

output reg [4:0] regA1\_O,

output reg [4:0] regA2\_O,

output reg [4:0] regA3\_O,

output reg [31:0] regRD1\_O,

output reg [31:0] regRD2\_O,

output reg [31:0] imm32\_O,

output reg [31:0] PCAdd8\_O,

output reg [31:0] PC\_O,

output reg memWrite\_O,

output reg [1:0] EXBackSel\_O,

output reg [1:0] MEMBackSel\_O,

output reg [1:0] WBBackSel\_O,

output reg ALUSrcASel\_O,

output reg ALUSrcBSel\_O,

output reg [3:0] ALUCtrl\_O,

output reg [2:0] DMCtrl\_O,

output reg [2:0] Tnew\_O

);

表14 ID\_EX端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | shamt\_I | I | 5 | 移位位数 |
| 5 | regA1\_I | I | 5 | rs |
| 6 | regA2\_I | I | 5 | rt |
| 7 | regA3\_I | I | 5 | GRF写入地址 |
| 8 | regRD1\_I | I | 32 | GRF[rs] |
| 9 | regRD2\_I | I | 32 | GRF[rt] |
| 10 | imm32\_I | I | 32 | 32位立即数 |
| 11 | PCAdd8\_I | I | 32 | PC+8 |
| 12 | PC\_I | I | 32 | 指令地址 |
| 13 | memWrite\_I | I | 1 | DM写使能信号 |
| 14 | EXBackSel\_I | I | 2 | 从E级转发的数据选择信号 |
| 15 | MEMBackSel\_I | I | 2 | 从M级转发的数据选择信号 |
| 16 | WBBackSel\_I | I | 2 | 从W级转发的数据选择信号 |
| 17 | ALUSrcASel\_I | I | 1 | ALU操作数A选择信号 |
| 18 | ALUSrcBSel\_I | I | 1 | ALU操作数B选择信号 |
| 19 | ALUCtrl\_I | I | 4 | ALU控制信号 |
| 20 | DMCtrl\_I | I | 3 | DM控制信号 |
| 21 | Tnew\_I | I | 3 | 当前指令的Tnew值 |
| 22 | shamt\_O | O | 5 | 移位位数 |
| 23 | regA1\_O | O | 5 | rs |
| 24 | regA2\_O | O | 5 | rt |
| 25 | regA3\_O | O | 5 | GRF写入地址 |
| 26 | regRD1\_O | O | 32 | GRF[rs] |
| 27 | regRD2\_O | O | 32 | GRF[rt] |
| 28 | imm32\_O | O | 32 | 32位立即数 |
| 29 | PCAdd8\_O | O | 32 | PC+8 |
| 30 | PC\_O | O | 32 | 指令地址 |
| 31 | memWrite\_O | O | 1 | DM写使能信号 |
| 32 | EXBackSel\_O | O | 2 | 从E级转发的数据选择信号 |
| 33 | MEMBackSel\_O | O | 2 | 从M级转发的数据选择信号 |
| 34 | WBBackSel\_O | O | 2 | 从W级转发的数据选择信号 |
| 35 | ALUSrcASel\_O | O | 1 | ALU操作数A选择信号 |
| 36 | ALUSrcBSel\_O | O | 1 | ALU操作数B选择信号 |
| 37 | ALUCtrl\_O | O | 4 | ALU控制信号 |
| 38 | DMCtrl\_O | O | 3 | DM控制信号 |
| 39 | Tnew\_O | O | 3 | 当前指令的Tnew值 |

内部逻辑说明：

为流水寄存器ID\_EX，在D级和E级间流水数据。

### EX\_MEM（Pipeline Register EX\_MEM）

模块定义：

module EX\_MEM(

input clk,

input reset,

input WE;

input [4:0] regA2\_I,

input [4:0] regA3\_I,

input [31:0] ALUResult\_I,

input [31:0] regRD2\_I,

input [31:0] PCAdd8\_I,

input [31:0] PC\_I,

input memWrite\_I,

input [1:0] MEMBackSel\_I,

input [1:0] WBBackSel\_I,

input [2:0] DMCtrl\_I,

input [2:0] Tnew\_I,

output reg [4:0] regA2\_O,

output reg [4:0] regA3\_O,

output reg [31:0] ALUResult\_O,

output reg [31:0] regRD2\_O,

output reg [31:0] PCAdd8\_O,

output reg [31:0] PC\_O,

output reg memWrite\_O,

output reg [1:0] MEMBackSel\_O,

output reg [1:0] WBBackSel\_O,

output reg [2:0] DMCtrl\_O,

output reg [2:0] Tnew\_O

);

表15 EX\_MEM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | regA2\_I | I | 5 | rt |
| 5 | regA3\_I | I | 5 | GRF写入地址 |
| 6 | ALUResult\_I | I | 32 | ALU运算结果 |
| 7 | regRD2\_I | I | 32 | GRF[rt] |
| 8 | PCAdd8\_I | I | 32 | PC+8 |
| 9 | PC\_I | I | 32 | 指令地址 |
| 10 | memWrite\_I | I | 1 | DM写使能信号 |
| 11 | MEMBackSel\_I | I | 2 | 从M级转发的数据选择信号 |
| 12 | WBBackSel\_I | I | 2 | 从W级转发的数据选择信号 |
| 13 | DMCtrl\_I | I | 3 | DM控制信号 |
| 14 | Tnew\_I | I | 3 | 当前指令的Tnew值 |
| 15 | regA2\_O | O | 5 | rt |
| 16 | regA3\_O | O | 5 | GRF写入地址 |
| 17 | ALUResult\_O | O | 32 | ALU运算结果 |
| 18 | regRD2\_O | O | 32 | GRF[rt] |
| 19 | PCAdd8\_O | O | 32 | PC+8 |
| 20 | PC\_O | O | 32 | 指令地址 |
| 21 | memWrite\_O | O | 1 | DM写使能信号 |
| 22 | MEMBackSel\_O | O | 2 | 从M级转发的数据选择信号 |
| 23 | WBBackSel\_O | O | 2 | 从W级转发的数据选择信号 |
| 24 | DMCtrl\_O | O | 3 | DM控制信号 |
| 25 | Tnew\_O | O | 3 | 当前指令的Tnew值 |

内部逻辑说明：

为流水寄存器EX\_MEM，在E级和M级间流水数据。

### MEM\_WB（Pipeline Register MEM\_WB）

模块定义：

module MEM\_WB(

input clk,

input reset,

input WE,

input [4:0] regA3\_I,

input [31:0] ALUResult\_I,

input [31:0] memRD\_I,

input [31:0] PCAdd8\_I,

input [31:0] PC\_I,

input [1:0] WBBackSel\_I,

input [2:0] Tnew\_I,

output reg [4:0] regA3\_O,

output reg [31:0] ALUResult\_O,

output reg [31:0] memRD\_O,

output reg [31:0] PCAdd8\_O,

output reg [31:0] PC\_O,

output reg [1:0] WBBackSel\_O,

output reg [2:0] Tnew\_O

);

表16 MEM\_WB端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | regA3\_I | I | 5 | GRF写入地址 |
| 5 | ALUResult\_I | I | 32 | ALU计算结果 |
| 6 | memRD\_I | I | 32 | DM数据输出 |
| 7 | PCAdd8\_I | I | 32 | PC+8 |
| 8 | PC\_I | I | 32 | 指令地址 |
| 9 | WBBackSel\_I | I | 2 | 从W级转发的数据选择信号 |
| 10 | Tnew\_I | I | 3 | 当前指令的Tnew值 |
| 11 | regA3\_O | O | 5 | GRF写入地址 |
| 12 | ALUResult\_O | O | 32 | ALU计算结果 |
| 13 | memRD\_O | O | 32 | DM数据输出 |
| 14 | PCAdd8\_O | O | 32 | PC+8 |
| 15 | PC\_O | O | 32 | 指令地址 |
| 16 | WBBackSel\_O | O | 2 | 从W级转发的数据选择信号 |
| 17 | Tnew\_O | O | 3 | 当前指令的Tnew值 |

内部逻辑说明：

为流水寄存器MEM\_WB，在M级和W级间流水数据。

### Datapath

模块定义：

module Datapath (

input clk,

input reset,

input memWrite\_D,

input [1:0] regAddrSel\_D,

input [1:0] EXBackSel\_D,

input [1:0] MEMBackSel\_D,

input [1:0] WBBackSel\_D,

input ALUSrcASel\_D,

input ALUSrcBSel\_D,

input [3:0] ALUCtrl\_D,

input [3:0] CMPCtrl\_D,

input [2:0] EXTCtrl\_D,

input [2:0] NPCCtrl\_D,

input [2:0] DMCtrl\_D,

input [2:0] Tnew\_D,

input [1:0] regRD1Forward\_D,

input [1:0] regRD2Forward\_D,

input [1:0] regRD1Forward\_E,

input [1:0] regRD2Forward\_E,

input regRD2Forward\_M,

input stall,

output [31:0] instr\_D,

output CMPResult\_D,

output [4:0] regA1\_D,

output [4:0] regA2\_D,

output [4:0] regA1\_E,

output [4:0] regA2\_E,

output [4:0] regA2\_M,

output [2:0] Tnew\_E,

output [2:0] Tnew\_M,

output [2:0] Tnew\_W,

output [4:0] regA3\_E,

output [4:0] regA3\_M,

output [4:0] regA3\_W

);

表17 Datapath端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 同步复位 |
| 3 | memWrite\_D | I | 2 | DM写使能信号 |
| 4 | regAddrSel\_D | I | 2 | GRF输入地址选择信号 |
| 5 | EXBackSel\_D | I | 2 | E级转发数据选择信号 |
| 6 | MEMBackSel\_D | I | 2 | M级转发数据选择信号 |
| 7 | WBBackSel\_D | I | 1 | W级转发数据选择信号 |
| 8 | ALUSrcASel\_D | I | 1 | ALU操作数A选择信号 |
| 9 | ALUSrcBSel\_D | I | 4 | ALU操作数B选择信号 |
| 10 | ALUCtrl\_D | I | 4 | ALU控制信号 |
| 11 | CMPCtrl\_D | I | 3 | CMP控制信号 |
| 12 | EXTCtrl\_D | I | 3 | EXT控制信号 |
| 13 | NPCCtrl\_D | I | 3 | NPC控制信号 |
| 14 | DMCtrl\_D | I | 3 | DM控制信号 |
| 15 | Tnew\_D | I | 3 | D级指令Tnew值 |
| 16 | regRD1Forward\_D | I | 2 | D级GRF[rs]转发数据选择信号 |
| 17 | regRD2Forward\_D | I | 2 | D级GRF[rt]转发数据选择信号 |
| 18 | regRD1Forward\_E | I | 2 | E级GRF[rs]转发数据选择信号 |
| 19 | regRD2Forward\_E | I | 2 | E级GRF[rt]转发数据选择信号 |
| 20 | regRD2Forward\_M | I | 1 | M级GRF[rt]转发数据选择信号 |
| 21 | stall | I | 1 | 阻塞信号 |
| 22 | instr\_D | O | 32 | 当前指令 |
| 23 | CMPResult\_D | O | 1 | CMP比较结果 |
| 24 | regA1\_D | O | 5 | D级rs |
| 25 | regA2\_D | O | 5 | D级rt |
| 26 | regA1\_E | O | 5 | E级rs |
| 27 | regA2\_E | O | 5 | E级rt |
| 28 | regA2\_M | O | 5 | M级rt |
| 29 | Tnew\_E | O | 3 | E级Tnew |
| 30 | Tnew\_M | O | 3 | M级Tnew |
| 31 | Tnew\_W | O | 3 | W级Tnew |
| 32 | regA3\_E | O | 5 | E级GRF写入地址 |
| 33 | regA3\_M | O | 5 | M级GRF写入地址 |
| 34 | regA3\_W | O | 5 | W级GRF写入地址 |

内部逻辑说明：

为数据通路，将IF、ID、EX、MEM等流水级，IF\_ID、ID\_EX、EX\_MEM、MEM\_WB等流水寄存器，以及数据转发的旁路等连接在一起。

### MainController

模块定义：

module MainController (

input [31:0] instr,

input flag,

output memWrite,

output [1:0] regAddrSel,

output [1:0] EXBackSel,

output [1:0] MEMBackSel,

output [1:0] WBBackSel,

output ALUSrcASel,

output ALUSrcBSel,

output [3:0] ALUCtrl,

output [3:0] CMPCtrl,

output [2:0] EXTCtrl,

output [2:0] NPCCtrl,

output [2:0] DMCtrl,

output [2:0] Tnew,

output [2:0] Tuse\_A1,

output [2:0] Tuse\_A2

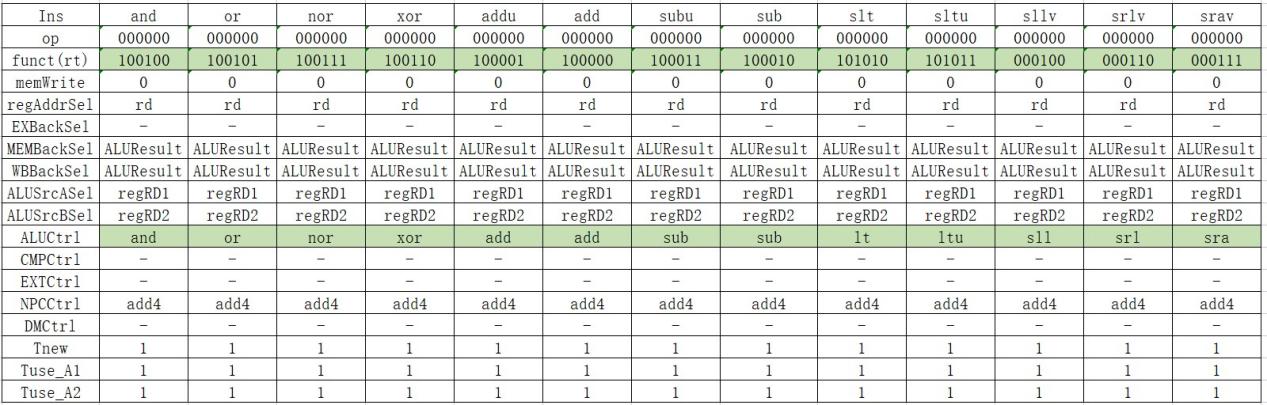
);

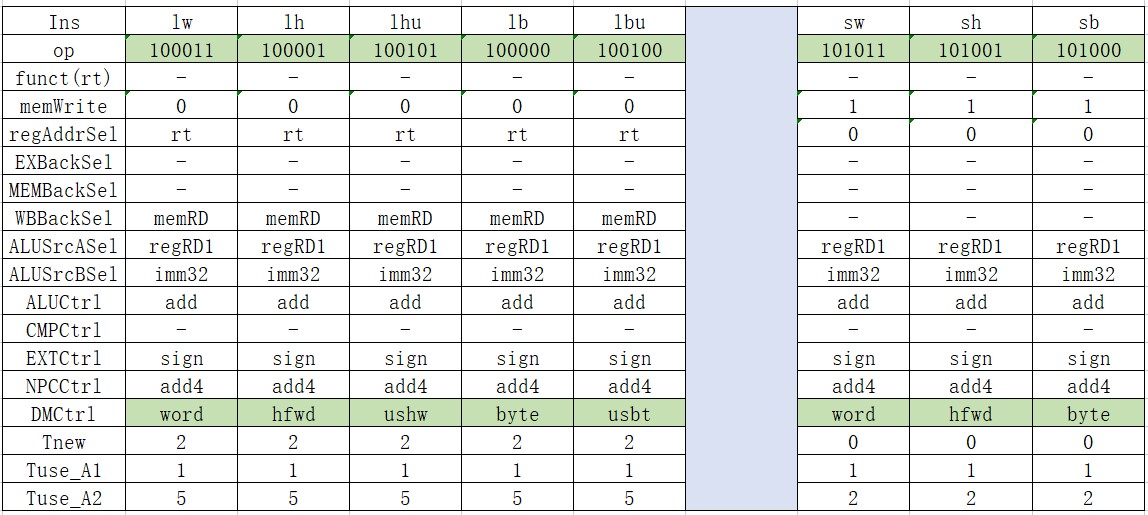
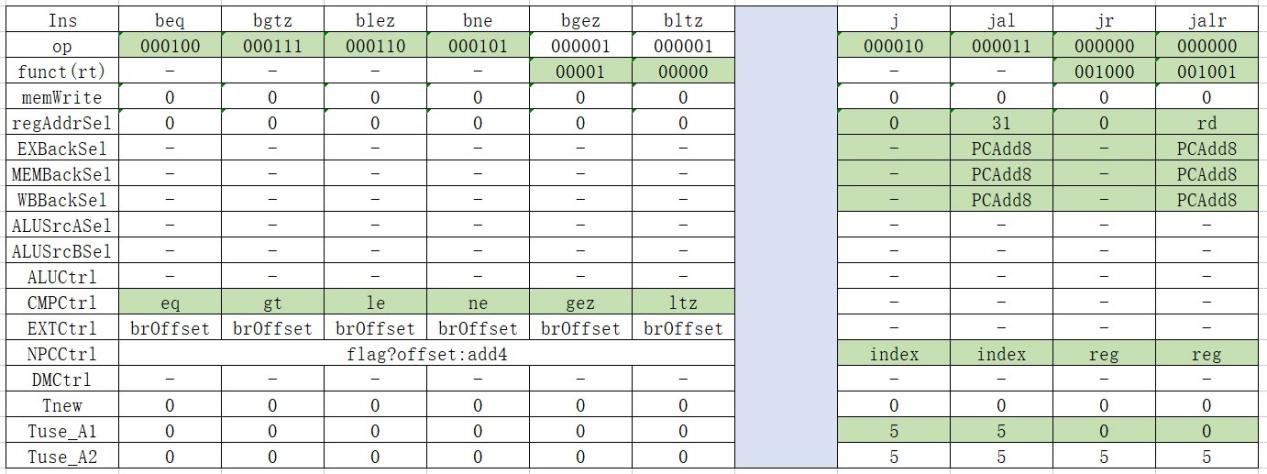
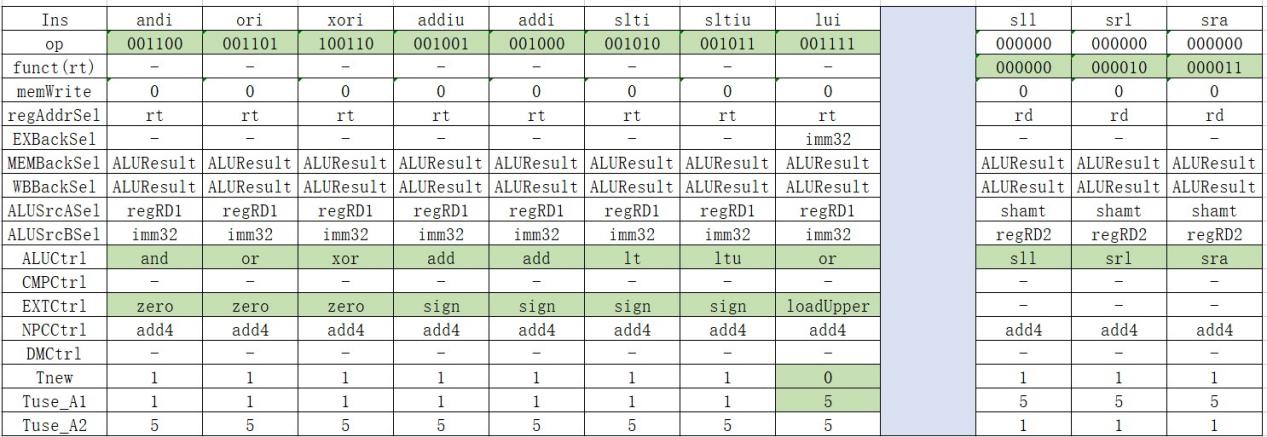
表18 MainController端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | instr | I | 32 | 当前指令 |
| 2 | flag | I | 1 | 跳转条件是否成立 |
| 3 | memWrite | O | 1 | DM写使能信号 |
| 4 | regAddrSel | O | 2 | GRF输入地址选择信号 |
| 5 | EXBackSel | O | 2 | E级转发数据选择信号 |
| 6 | MEMBackSel | O | 2 | M级转发数据选择信号 |
| 7 | WBBackSel | O | 2 | W级转发数据选择信号 |
| 8 | ALUSrcASel | O | 1 | ALU操作数A选择信号 |
| 9 | ALUSrcBSel | O | 1 | ALU操作数B选择信号 |
| 10 | ALUCtrl | O | 4 | ALU控制信号 |
| 11 | CMPCtrl | O | 4 | CMP控制信号 |
| 12 | EXTCtrl | O | 3 | EXT控制信号 |
| 13 | NPCCtrl | O | 3 | NPC控制信号 |
| 14 | DMCtrl | O | 3 | DM控制信号 |
| 15 | Tnew | O | 3 | 当前指令Tnew值 |
| 16 | Tuse\_A1 | O | 3 | 当前指令rs的Tuse值 |
| 17 | Tuse\_A2 | O | 3 | 当前指令rt的Tuse值 |

内部逻辑说明：

为主控制器，用于指令的识别和部分控制信号的生成。移码方式选择控制信号驱动型，为了防止代码膨胀，采用聚合连线的方式。



图1 指令与控制信号对照表

### HazardSolveUnit

模块定义：

module HazardSolveUnit (

input [2:0] Tuse\_A1\_D,

input [2:0] Tuse\_A2\_D,

input [4:0] regA1\_D,

input [4:0] regA2\_D,

input [4:0] regA1\_E,

input [4:0] regA2\_E,

input [4:0] regA2\_M,

input [2:0] Tnew\_E,

input [2:0] Tnew\_M,

input [2:0] Tnew\_W,

input [4:0] regA3\_E,

input [4:0] regA3\_M,

input [4:0] regA3\_W,

output [1:0] regRD1Forward\_D,

output [1:0] regRD2Forward\_D,

output [1:0] regRD1Forward\_E,

output [1:0] regRD2Forward\_E,

output regRD2Forward\_M,

output stall

);

表19 HazardSolveUnit端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | Tuse\_A1\_D | I | 3 | D级rs的Tuse值 |
| 2 | Tuse\_A2\_D | I | 3 | D级rt的Tuse值 |
| 3 | regA1\_D | I | 5 | D级rs |
| 4 | regA2\_D | I | 5 | D级rt |
| 5 | regA1\_E | I | 5 | E级rs |
| 6 | regA2\_E | I | 5 | E级rt |
| 7 | regA2\_M | I | 5 | M级rt |
| 8 | Tnew\_E | I | 3 | E级Tnew |
| 9 | Tnew\_M | I | 3 | M级Tnew |
| 10 | Tnew\_W | I | 3 | W级Tnew |
| 11 | regA3\_E | I | 5 | E级GRF写入地址 |
| 12 | regA3\_M | I | 5 | M级GRF写入地址 |
| 13 | regA3\_W | I | 5 | W级GRF写入地址 |
| 14 | regRD1Forward\_D | O | 2 | D级GRF[rs]转发数据选择信号 |
| 15 | regRD2Forward\_D | O | 2 | D级GRF[rt]转发数据选择信号 |
| 16 | regRD1Forward\_E | O | 2 | E级GRF[rs]转发数据选择信号 |
| 17 | regRD2Forward\_E | O | 2 | E级GRF[rt]转发数据选择信号 |
| 18 | regRD2Forward\_M | O | 1 | M级GRF[rt]转发数据选择信号 |
| 19 | stall | O | 1 | 阻塞信号 |

内部逻辑说明：

为冲突处理单元，用于生成转发或阻塞的控制信号。当先执行指令的目的寄存器不为0且与后执行指令的源寄存器相匹配时，进行转发。当先执行指令的Tnew值小于后执行指令的Tuse值且寄存器相匹配且不为0时，进行暴力阻塞。

### Controller

模块定义：

module Controller (

input [31:0] instr,

input CMPResult,

input [4:0] regA1\_D,

input [4:0] regA2\_D,

input [4:0] regA1\_E,

input [4:0] regA2\_E,

input [4:0] regA2\_M,

input [2:0] Tnew\_E,

input [2:0] Tnew\_M,

input [2:0] Tnew\_W,

input [4:0] regA3\_E,

input [4:0] regA3\_M,

input [4:0] regA3\_W,

output memWrite,

output [1:0] regAddrSel,

output [1:0] EXBackSel,

output [1:0] MEMBackSel,

output [1:0] WBBackSel,

output ALUSrcASel,

output ALUSrcBSel,

output [3:0] ALUCtrl,

output [3:0] CMPCtrl,

output [2:0] EXTCtrl,

output [2:0] NPCCtrl,

output [2:0] DMCtrl,

output [2:0] Tnew,

output [1:0] regRD1Forward\_D,

output [1:0] regRD2Forward\_D,

output [1:0] regRD1Forward\_E,

output [1:0] regRD2Forward\_E,

output regRD2Forward\_M,

output stall

);

表20 Controller端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | instr | I | 32 | 当前指令 |
| 2 | CMPResult | I | 1 | CMP比较结果 |
| 3 | regA1\_D | I | 5 | D级rs |
| 4 | regA2\_D | I | 5 | D级rt |
| 5 | regA1\_E | I | 5 | E级rs |
| 6 | regA2\_E | I | 5 | E级rt |
| 7 | regA2\_M | I | 5 | M级rt |
| 8 | Tnew\_E | I | 3 | E级Tnew |
| 9 | Tnew\_M | I | 3 | M级Tnew |
| 10 | Tnew\_W | I | 3 | W级Tnew |
| 11 | regA3\_E | I | 5 | E级GRF写入地址 |
| 12 | regA3\_M | I | 5 | M级GRF写入地址 |
| 13 | regA3\_W | I | 5 | W级GRF写入地址 |
| 14 | memWrite | O | 2 | DM写使能信号 |
| 15 | regAddrSel | O | 2 | GRF输入地址选择信号 |
| 16 | EXBackSel | O | 2 | E级转发数据选择信号 |
| 17 | MEMBackSel | O | 2 | M级转发数据选择信号 |
| 18 | WBBackSel | O | 1 | W级转发数据选择信号 |
| 19 | ALUSrcASel | O | 1 | ALU操作数A选择信号 |
| 20 | ALUSrcBSel | O | 4 | ALU操作数B选择信号 |
| 21 | ALUCtrl | O | 4 | ALU控制信号 |
| 22 | CMPCtrl | O | 3 | CMP控制信号 |
| 23 | EXTCtrl | O | 3 | EXT控制信号 |
| 24 | NPCCtrl | O | 3 | NPC控制信号 |
| 25 | DMCtrl | O | 3 | DM控制信号 |
| 26 | Tnew | O | 3 | D级指令Tnew值 |
| 27 | regRD1Forward\_D | O | 2 | D级GRF[rs]转发数据选择信号 |
| 28 | regRD2Forward\_D | O | 2 | D级GRF[rt]转发数据选择信号 |
| 29 | regRD1Forward\_E | O | 2 | E级GRF[rs]转发数据选择信号 |
| 30 | regRD2Forward\_E | O | 2 | E级GRF[rt]转发数据选择信号 |
| 31 | regRD2Forward\_M | O | 1 | M级GRF[rt]转发数据选择信号 |
| 32 | stall | O | 1 | 阻塞信号 |

内部逻辑说明：

为控制器，连接MainController和HazardSolveUnit。从Datapath中获得数据，并向其发送控制信号。

### （三）重要机制实现方法

#### 1. 跳转

NPC模块、EXT模块、ALU模块协同工作支持指令beq的跳转机制。

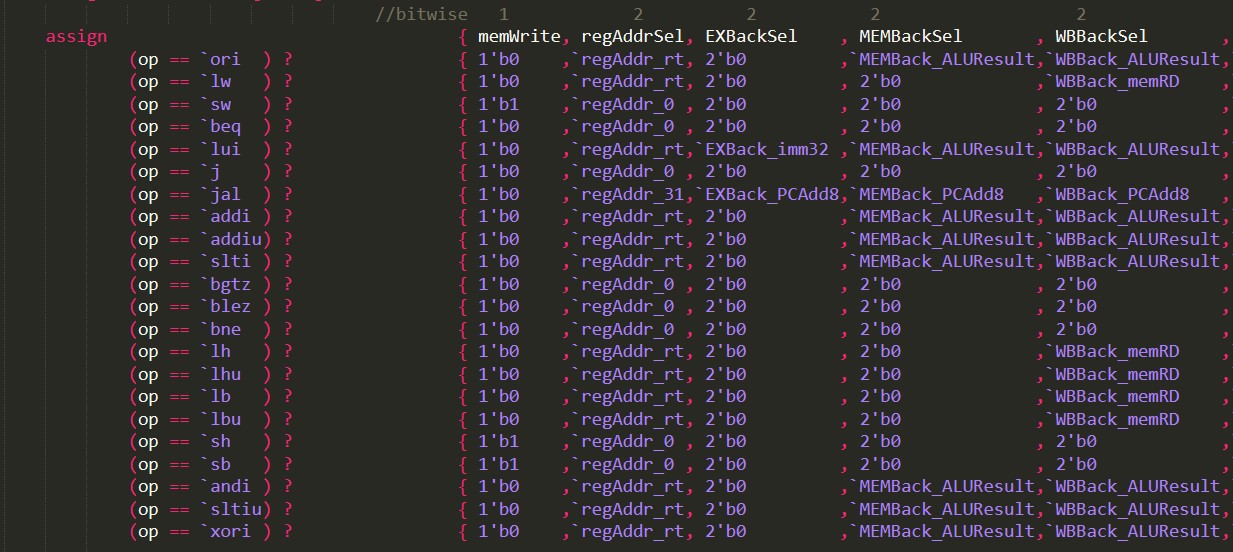
NPC模块内置了判定单元和计算单元来独立支持指令j、jal、jr 的跳转机制。

#### 2. 半字、字节存取

通过控制信号DMCtrl来判断是对字、半字还是字节进行操作。对于写指令，直接根据地址信号，找到相应位置的字、半字或字节进行写入。对于读指令，先根据地址取出相应位置的字、半字或字节，再根据控制信号进行零扩展或符号扩展后输出。

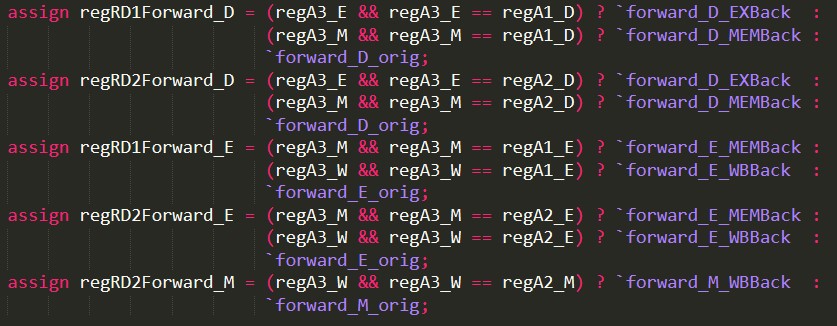
#### 3. 主控制器

采用集中式译码和指令驱动型译码。为了防止代码膨胀，使用assign而不是always和阻塞赋值来生成控制信号，并利用宏区分信号类别。

图2 控制信号生成的部分代码

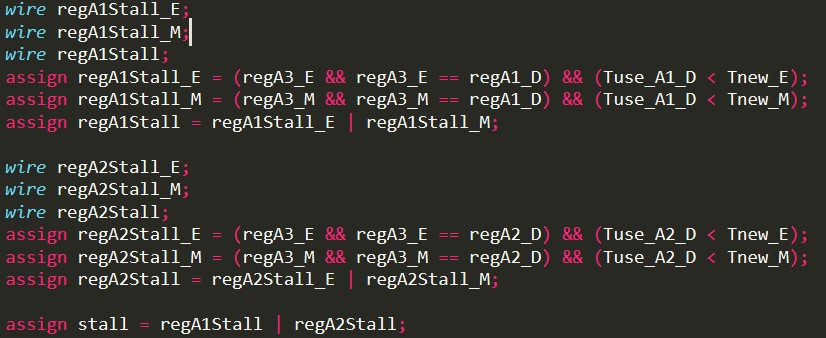
#### 4. 转发机制

当前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为 0，就选择该转发输入来源，在有多个转发输入来源都满足条件时，最新产生的数据优先级最高。采取暴力转发的方式，即不需要判断指令间的Tuse和Tnew的关系，因为当条件不成立时会引发阻塞，而阻塞的优先级更高。

图3 转发控制信号的部分代码

#### 阻塞机制

当 D 级指令读取寄存器的地址与 E 级或 M 级的指令写入寄存器的地址相等且不为 0，且 D 级指令的 Tuse 小于对应 E 级或 M 级指令的 Tnew 时，在 D 级暂停指令。阻塞时将PC与IF\_ID寄存器的写使能信号赋为0，并且刷新ID\_EX寄存器。

图4 阻塞控制信号的部分代码

## 二、测试方案

### （一）典型测试样例

对于自动生成的样例见自动测试工具和思考题第六题。

手造的样例主要是用于补充自动生成样例中不足的地方：

ori $t0, $0, 123

addu $a0, $t0, $0

lw $a0, -123($a0)

lui $t3, 123

lui $t3, 0

lw $t4, 0($t3)

ori $t0, $0, 0x3028

addu $a0, $t0, $0

jr $a0

ori $t0, $0, 0x3038

sw $t0, 0($0)

lw $a0, 0($0)

jr $a0

nop

jal f

nop

j end

nop

f: jr $ra

nop

end:

jal tag

nop

tag: lw $t0, -0x3058($ra)

lw $s0, -0x3038($t0)

lw $s1, -0x3038($s0)

lw $s2, -0x3038($s1)

sw $t0, -0x3038($s2)

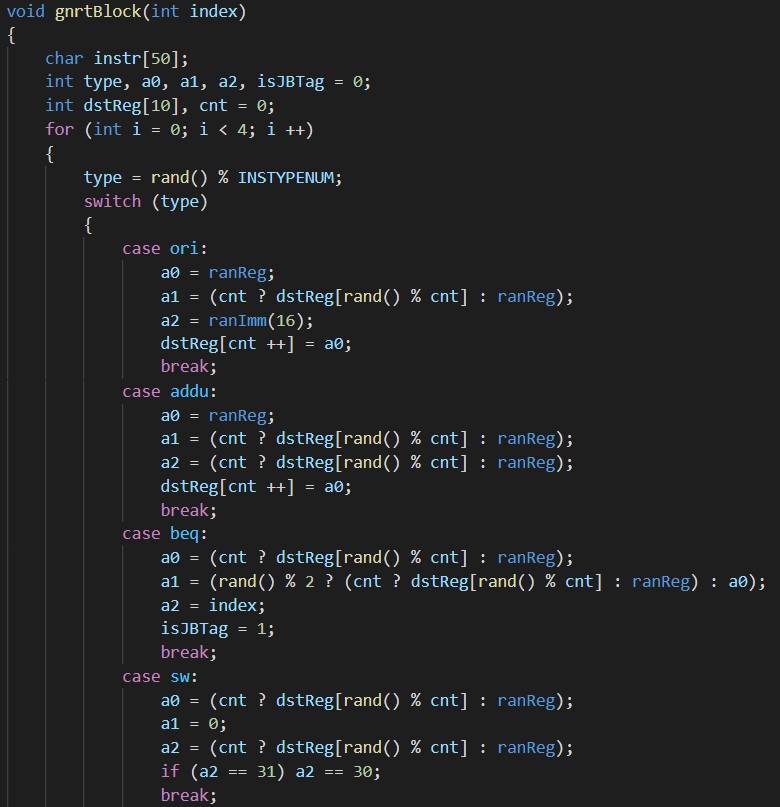
### （二）自动测试工具

#### 1. 测试样例生成器

运行环境：win10 g++ 11.1.0

程序大致流程为：

1. 利用ori和sw初始化寄存器和部分内存。
2. 将随机四条指令加一个标签组成一个代码块，保证代码块中后面指令的源寄存器是前面指令的目的寄存器，并且所有跳转标签为这个代码块后的标签。按以上规则随机生成若干个代码块。
3. 进行延迟槽检查。当延迟槽内出现跳转指令时，在前面插入一条无关指令
4. 进行写入地址和跳转地址检查。通过插入指令修改寄存器的值来保证地址正确。

图5 测试样例生成器部分代码

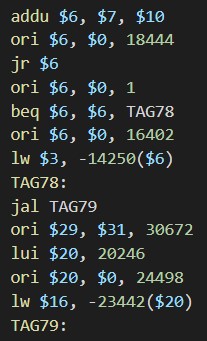


图6 生成的部分测试样例

#### 自动执行脚本

注：将P4的自动执行脚本做略微改动即可使用。所做的改动有：mars中代码段上限改为0x00006ffc、命令行运行mars时加入延迟槽和执行指令数上限、修改verilog的工程路径。

运行环境：win10 64位 python 3.9.6

步骤1：爆改mars，加入格式化输出，并把$gp和$sp的初始值改为0，把代码段的上限改为0x00006ffc。

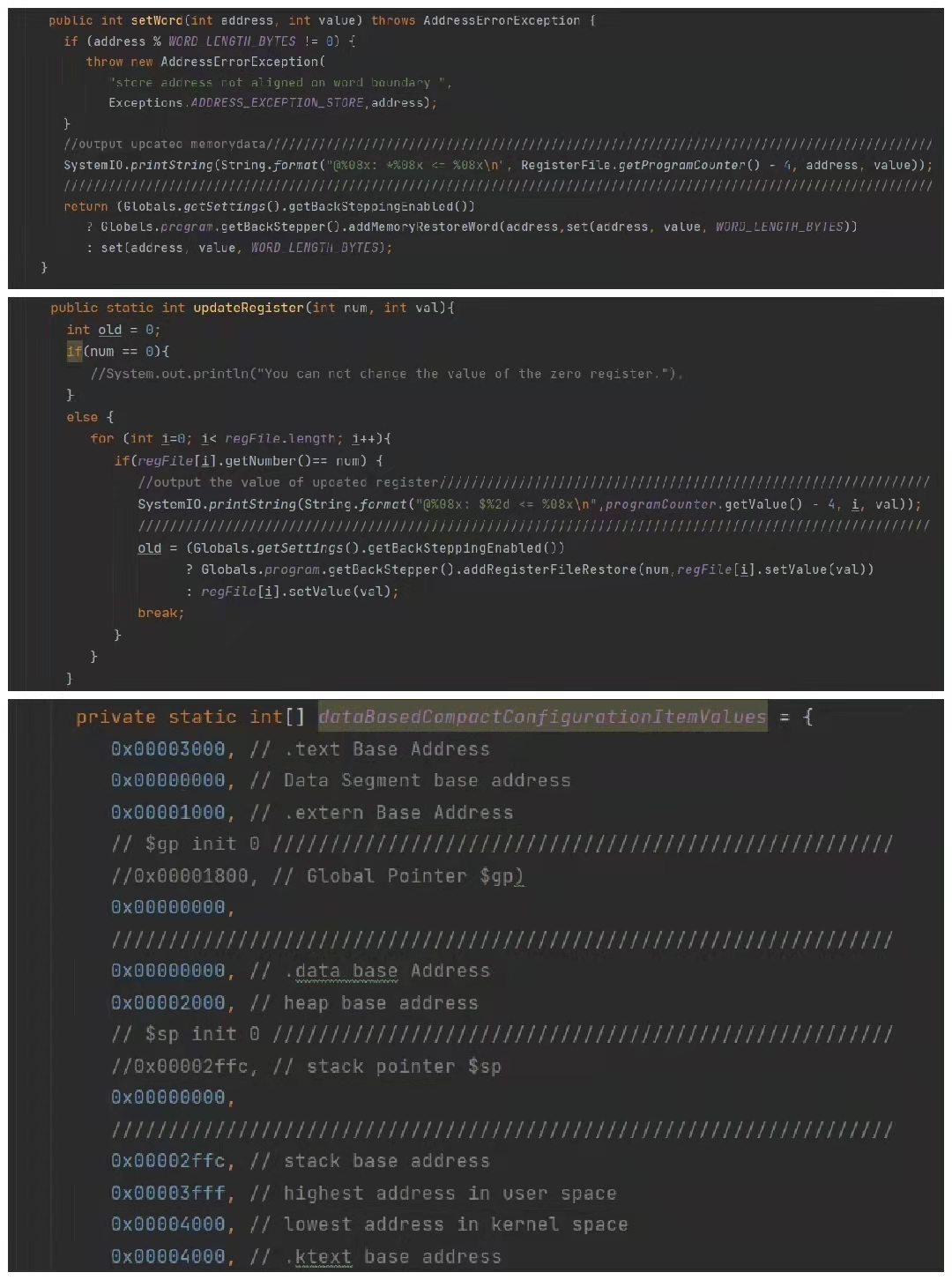
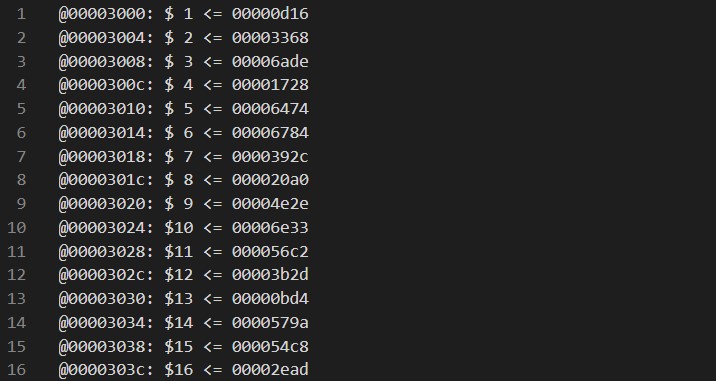


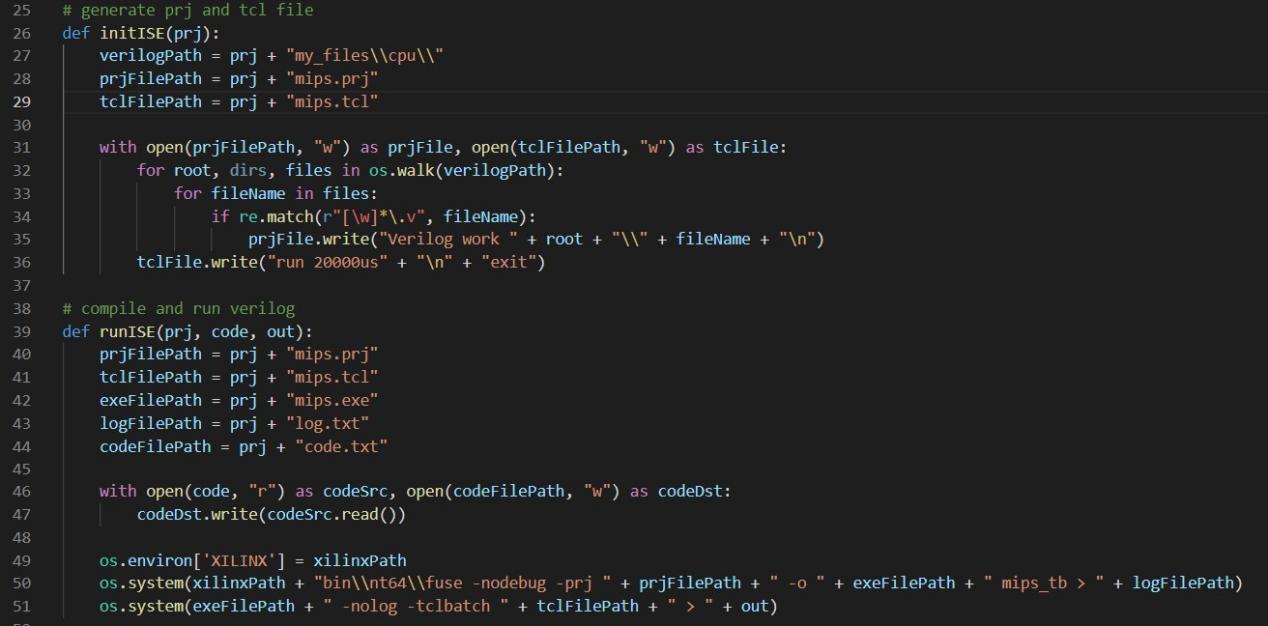
图7 修改的mars代码

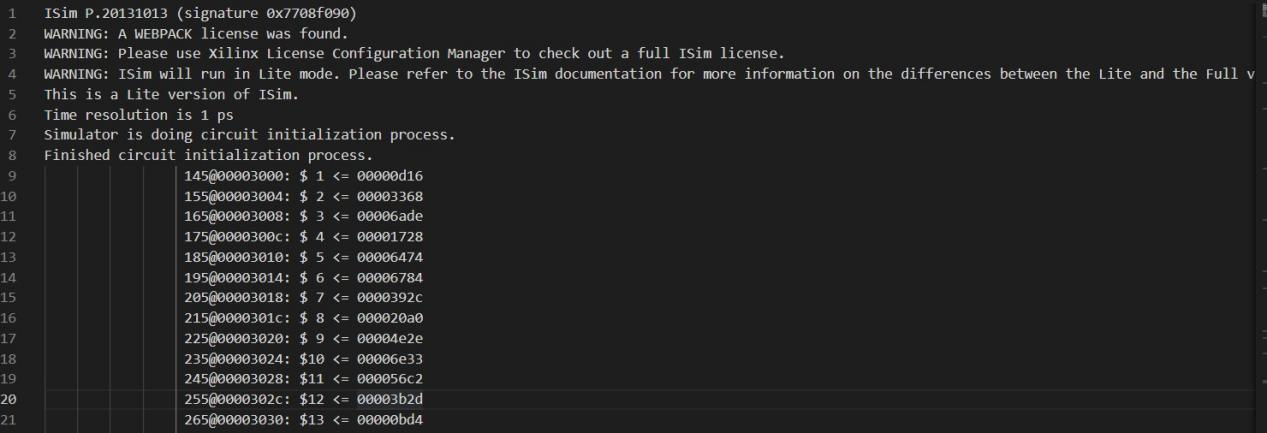
步骤2：运行mars，将结果输出到文件，并导出指令

图8 运行mars和导出指令的脚本

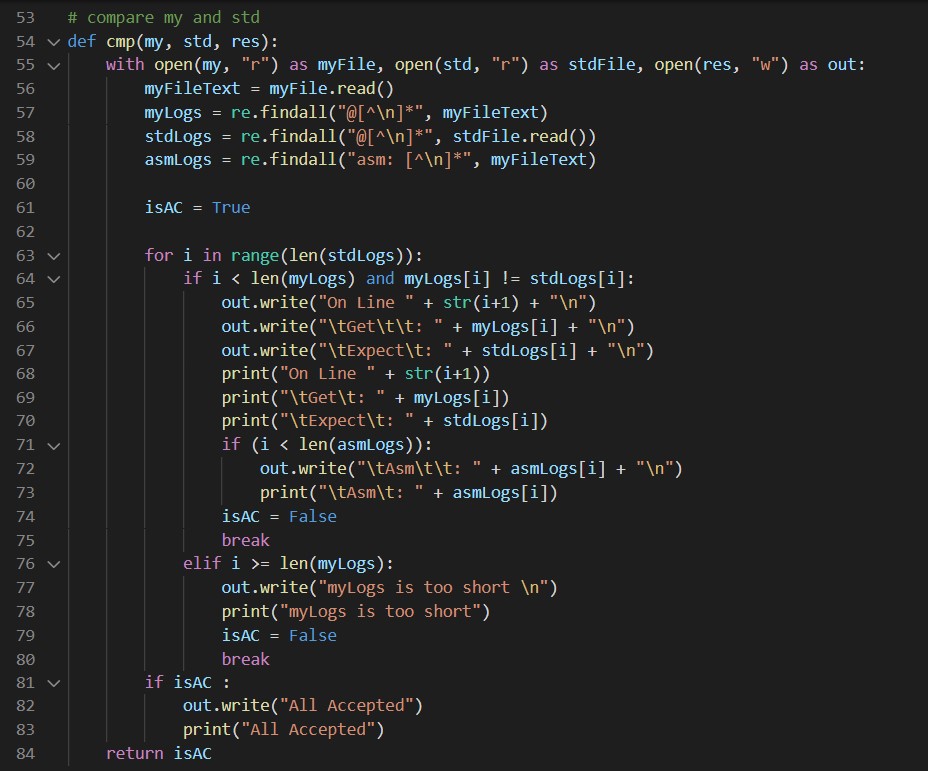
图9 mars的运行结果

步骤3：生成prj和tcl文件，编译verilog文件，进行仿真并将结果输出到文件

图10 ISE运行脚本

图11 ISE的运行结果

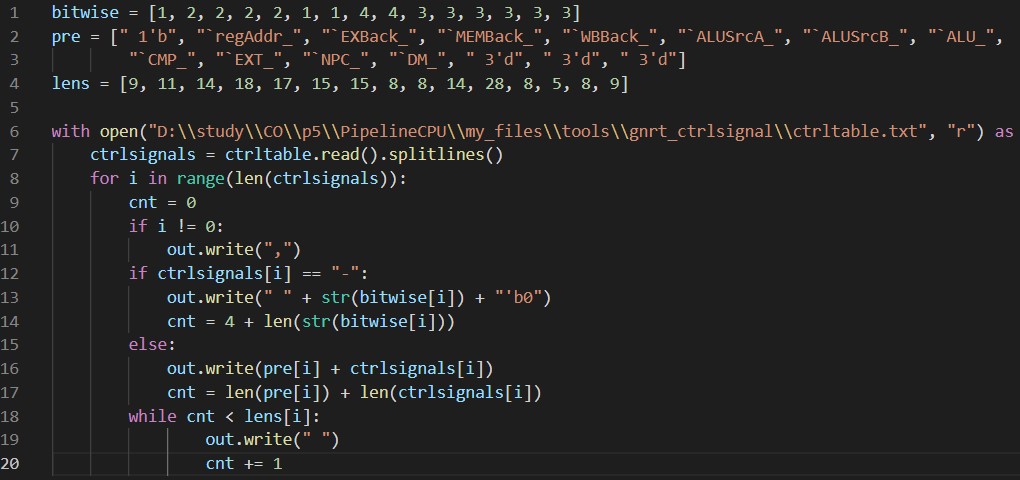
步骤4：将mars和ISE的运行结果进行文本比对，如果出错则给出错误信息

图12 文本比对程序

### （三）其他自动化工具

#### 1. 自动生成控制信号：

将指令与控制信号对应表中的某一列复制到ctrltable.txt文件中，再运行这个程序就可以得到一行格式化的控制信号，直接复制到MainController即可完成新增指令的控制信号生成。

图13 生成控制信号程序

#### 2.自动生成模块端口：

像Controller、Datapath、流水寄存器的模块的端口高达三四十个。将模块定义时的端口声明复制到iotable.txt中，运行该程序，即可得到一列实例化时用到的.pinName(pinName)格式的代码，复制到要实例化模块的地方即可。

图14 生成模块端口程序

## 三、思考题

### （一）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。

对于数据通路，由于IF\_ID流水寄存器和延迟槽的存在，应分别将F级的PC和D级的PC传入NPC，F级的PC用于计算正常的PC+4，而D级的PC用于计算跳转地址和输出PC+8。正常情况下，PC每个周期加4，但当跳转条件满足时（包括无条件跳转），NPC会计算出相应的地址，在下个周期跳转到这个地址。

对于控制信号，在D级指令译码阶段，控制器根据指令会生成NPC控制信号，从而选择下一个PC值。当beq或jr发生数据冲突时，控制器还会生成转发或阻塞的控制信号。

### （二）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 ？

因为有延迟槽的存在，所以需要链接延迟槽后一条指令，否则跳转回来时又会执行一遍延迟槽。

### 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？

ALU或DM等部件延时较长，如果由ALU或DM等部件提供数据，将大大增加关键路径所需时间，使流水线各阶段延迟不均衡，时钟周期增加，性能严重下降。比方说ALU需要来自DM产生的操作数，如果从MEM\_WB流水寄存器转发回来，那么关键路径只有ALU；但如果直接从DM转发回来，那么关键路劲至少包含DM+ALU，而且还可能有DM前的旁路，这样时钟周期可能会是原来的两倍甚至更多。

### “转发（旁路）机制的构造”中的 Thinking 1-4;

### Thinking1：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。

如果原始数据本来是错误的，并且等到使用它时转发来源已经离开流水线，那么就会导致数据错误。

比如指令序列：

add $t0, $t1, $t2

nop

sw $t0, 0($s0)

当sw在D级、add在M级时，ALU的数据已经准备好可以转发给D级的$t0，但是sw在D级时还不需要用$t0，如果sw还继续采用原来的$t0，那么等到M级需要用时，W级此时是nop，转发来源已经消失，sw存储的数据就是错误的。

### Thinking2：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？

若不采用GRF内部转发机制，当A1（或A2）与A3相同时，GRF可能输出的值就有两种可能，一种是输出寄存器原本的值，另一种是输出准备写入的数据。所以采用内部转发机制可以防止输出结果发生变化或者不确定。而且不用再考虑外部转发，节省了部分资源，更加易于设计和实现。

如果不采用内部转发机制，可以通过像E->D，M->D那样构造旁路并增加控制信号来实现外部转发。

### Thinking3：为什么 0 号寄存器需要特殊处理？

写入0号寄存器相当于不写入，也就是并不会改变0号寄存器原有的值，如果前面的指令向0寄存器写入非零值，就会转发回来错误的数据。

### Thinking4：什么是“最新产生的数据”？

“最新产生的数据”就是最后一条指令写入的数据，按照“新鲜”程度排序的话，E级>M级>W级，所以能从E级转发的就不从M和W级转发，能从M级转发就不从W级转发。

### 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？

AT法的核心思想是尽可能只通过A和T来生成转发和阻塞的控制信号。因为写入0寄存器等于不进行写入，所以可以把GRF的写使能信号包含在A中，当不写入寄存器或是写入0号寄存器时，A可以都设为0，就不需要额外传入WE了。

在翻译出A时可以进行特殊处理，如果使能信号WE为0就将A设成0。但笔者并没有采用这种做法，而是“忽略”使能信号，将其始终设为1，只通过写入地址是否为0来判断是否要进行写入。可以在写入地址的多路选择器上增加一个输入0，当指令不写入寄存器时，控制器产生的控制信号可以让该多路选择器选择0地址，这样在GRF和冲突处理单元中只需要判断写入地址，在GRF中也不需要在翻译A时做特殊处理了。

### 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？

考虑P5的MIPS-lite2 指令集，可以划分为一下几个类：寄存器立即数计算CalRI（以ori为代表，后面同理），双寄存器计算CalRR（addu）， 双寄存器分支BrRR（beq），写内存Store（sw），读内存Load（lw），跳转并链接jal，跳转寄存器jr，加载高位lui。（不考虑测试nop、j，因为并不会引起冲突）

按转发的方向或阻塞的原因分类，可能产生的冲突类型有（同时给出样例）：

转发：

1. E->D:

lui $t0, 123

addu $t1, $t1, $t0

1. M->D:

addu $t1, $t0, $t0

nop

addu $t2, $t1, $t1

1. W->D:

lw $t1, 0($0)

nop

nop

addu $t2, $t1, $t1

1. M->E:

addu $t1, $t0, $t0

addu $t2, $t1, $t1

1. W->E:

addu $t1, $t0, $t0

nop

addu $t2, $t1, $t1

1. W->M

addu $t1, $t0, $t0

sw $t1, 0($0)

阻塞：

1. E级导致的阻塞：

addu $t0, $t1, $t1

beq $t0, $t1, TAG

1. M级导致的阻塞：

lw $t0, 0($t1)

nop

beq $t0, $t1, TAG

对于W->D的转发，可通过GRF的内部转发解决，其他的转发需要另开旁路，并通过多路选择器在转发数据来源端和使用端分别进行数据选择。而对于不可转发的冲突，使用AT法暴力阻塞即可。

笔者的样例生成程序主要方法是：先随机生成四条指令一组的指令块，在生成的时候，保证后面指令的源寄存器一定是前面指令的目的寄存器，从而构成连续数据冒险序列。并且之后会进行指令合法性检查，比如在连续的跳转指令间插入无关指令、修改访存的地址、修改跳转寄存器的值等。

程序的不足主要有两方面：一是在进行合法性检查时，由于没能想到特别好的方法，所以主要是通过插入指令的方式修改相关的值，因此会破坏本来的连续冒险序列。二是目前只从每个指令类中取一个代表性指令进行测试，并没有实现所有相同类型的其他指令的测试。

考虑到程序有这些不足的地方，又根据样例分析程序给出的结果手动造了一组数据，用于弥补自动生成程序遗漏之处。最后经分析forward和stall的均分分别能达到85和95，并且没有warning。