# 计算机组成原理P8 MIPS微系统实验报告

20373944 何天然

## 一、设计方案综述

### （一）总体设计概述

本微系统包含了Verilog实现的流水线MIPS-CPU、系统桥、计时器、外设控制器等，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、mult、multu、div、divu、mfhi、mflo、mthi、mtlo、mfc0、mtc0、eret }共53条，支持AdEL、AdES、RI、Ov等异常、外设中断和读写。为了实现这些功能，CPU主要包含 CPU、Clock、Bridge、IM、DM、TC、DT、IO、UART、CP0、Controller、MainController、HazardSolveUnit、Datapath、IF、IF\_ID、ID、ID\_EX、EX、EX\_MEM、MEM、MEM\_WB、PC、NPC、EXT、CMP、GRF、ALU、MDU、BE、DE，这些模块共可分成四层，顶层为CPU、Clock、Bridge、TC、DT、IO、UART，其中CPU包含Controller和 Datapath；Controller包含MainController和HazardSolveUnit；Datapath包含IF、ID、EX、MEM、IF\_ID、ID\_EX、EX\_MEM、MEM\_WB、CP0，其中IF包含PC，ID包含NPC、EXT、CMP、GRF，EX包含ALU、MDU，MEM包含BE、DE。

### 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. MDU（MultiplyDivideUnit）

模块定义：

module MDU (

    input clk,

    input reset,

    input [31:0] srcA,

    input [31:0] srcB,

    input start,

    input [3:0] ctrl,

    output busy,

    output reg [31:0] HI,

    output reg [31:0] LO

);

表3 MDU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 | 同步复位信号 |
| 3 | srcA | I | 32 | 操作数A |
| 4 | srcB | I | 32 | 操作数B |
| 5 | start | I | 1 | 开始信号 |
| 6 | ctrl | I | 4 | 控制信号 |
| 7 | busy | O | 1 | 忙碌信号 |
| 8 | HI | O | 32 | HI寄存器的值 |
| 9 | LO | O | 32 | LO寄存器的值 |

内部逻辑说明：

可进行乘除法运算和mf、mt指令，内部采用状态机来实现乘除槽。

### 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. BE（ByteEnable）

模块定义：

module BE(

input [31:0] WD\_orig,

    input [1:0] byteSel,

    input [2:0] ctrl,

    output [3:0] byteEn,

    output [31:0] WD

);

表6 BE端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | WD\_orig | I | 32 | 原始写入数据 |
| 2 | byteSel | I | 4 | 字节选择信号 |
| 3 | ctrl | I | 3 | 控制信号 |
| 4 | byteEn | I | 4 | 字节使能信号 |
| 5 | WD | I | 32 | 写入数据 |

内部逻辑说明：

根据控制信号和字节选择信号，生成字节使能信号，并对写入数据进行处理。

### DE（DataExtender）

模块定义：

module DE(

input [31:0] RD\_orig,

    input [1:0] byteSel,

    input [2:0] ctrl,

    output [31:0] RD

);

表7 DE端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | RD\_orig | I | 32 | 原始读取数据 |
| 2 | byteSel | I | 2 | 字节选择信号 |
| 3 | ctrl | I | 3 | 控制信号 |
| 4 | RD | O | 32 | 读取数据 |

内部逻辑说明：

根据字节选择信号和控制信号，对原始读取数据进行位扩展等处理，然后输出正确的读取数据。

### 8. EXT（Extender）

模块定义：

module EXT(

input [15:0] imm16,

input [2:0] ctrl,

output [31:0] imm32

);

表8 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。

### 9. CMP（Comparer）

模块定义：

module ALU(

input [31:0] srcA,

input [31:0] srcB,

input [3:0] ctrl,

output result

);

表9 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10 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11 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12 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13 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14 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15 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16 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17 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18 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

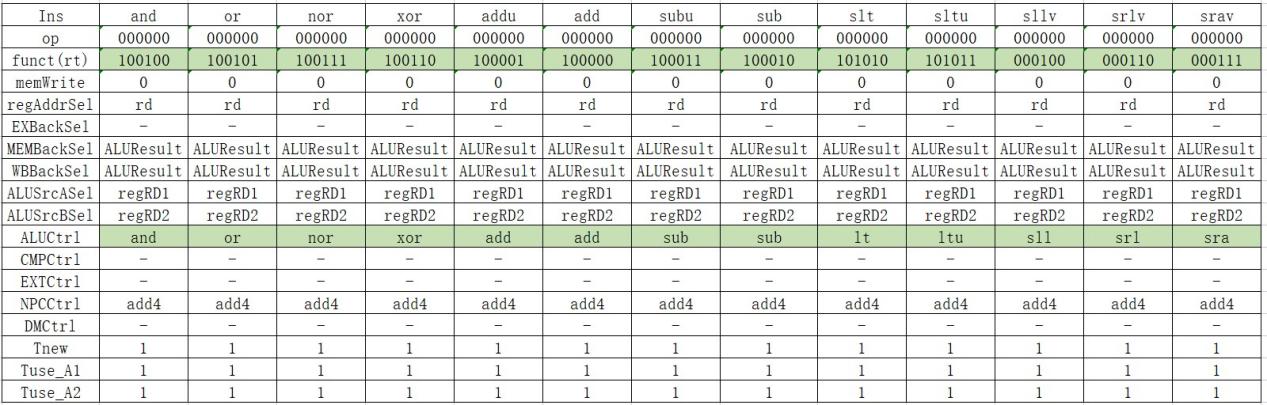
);

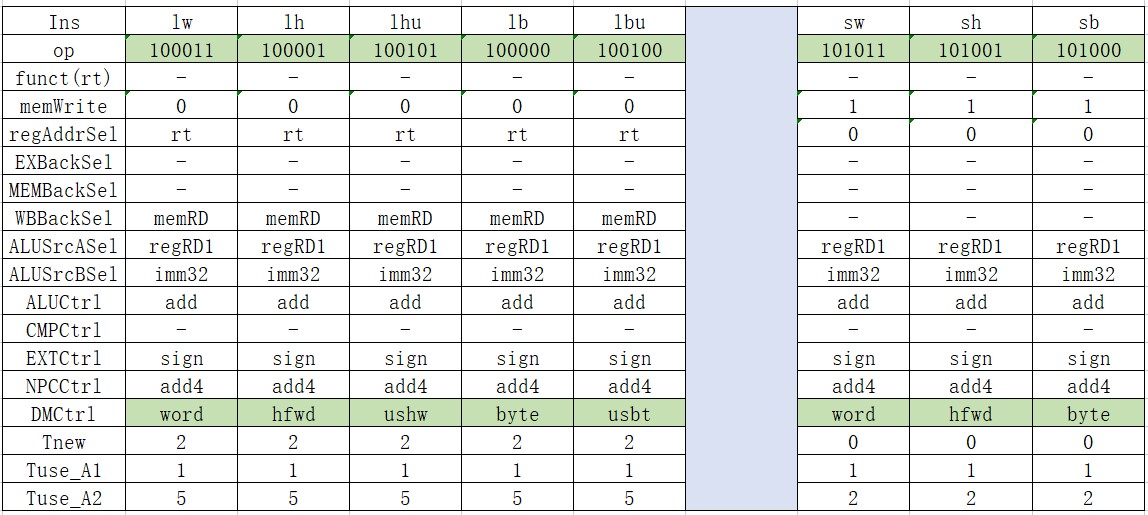
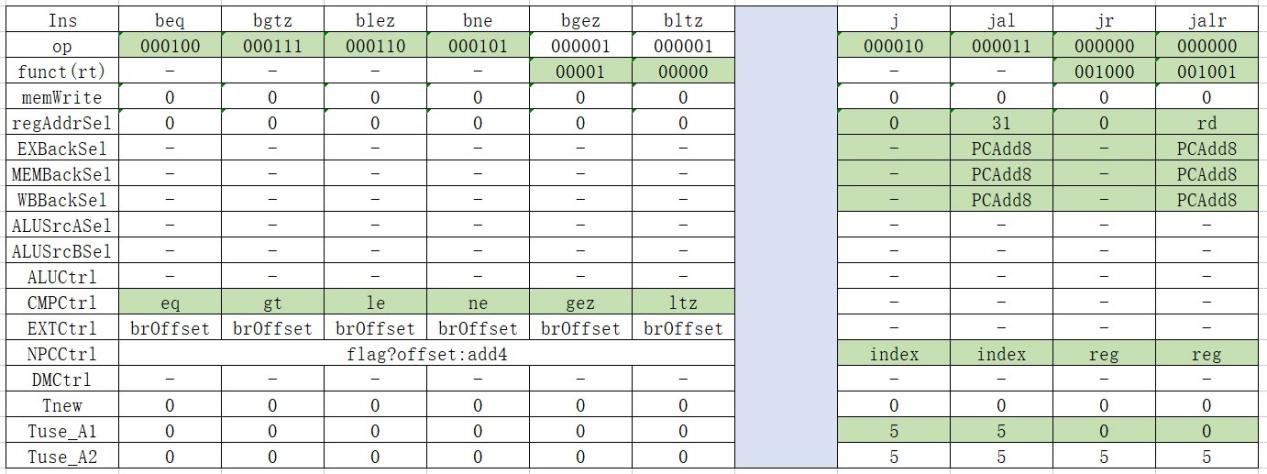
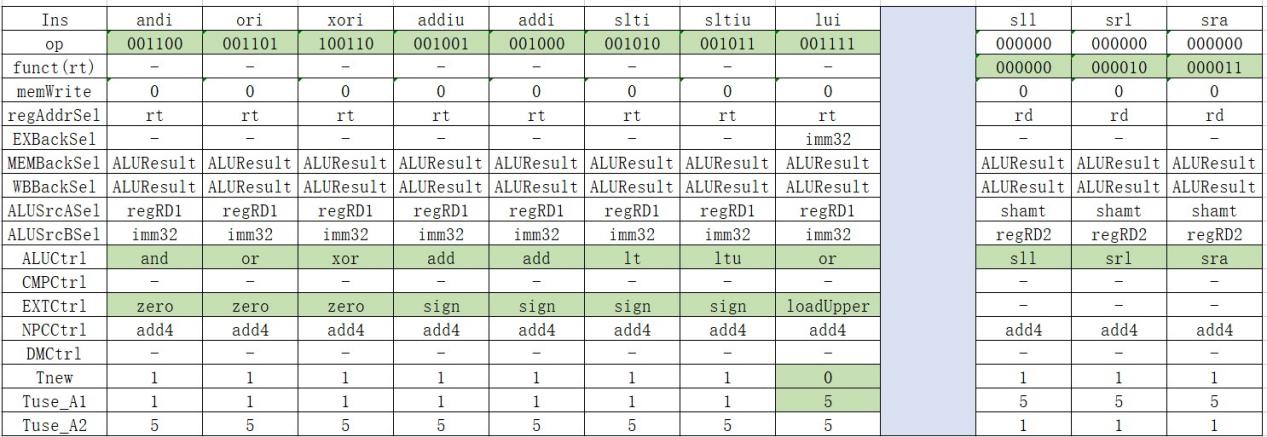
表19 MainController端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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20 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21 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。

### CPU：见P8部分

### CP0: 见（四）

### Bridge: 见（四）

### TC: 见计时器说明文档

### （三）重要机制实现方法

#### 1. 跳转

NPC模块、EXT模块、CMP模块协同工作支持指令beq的跳转机制。

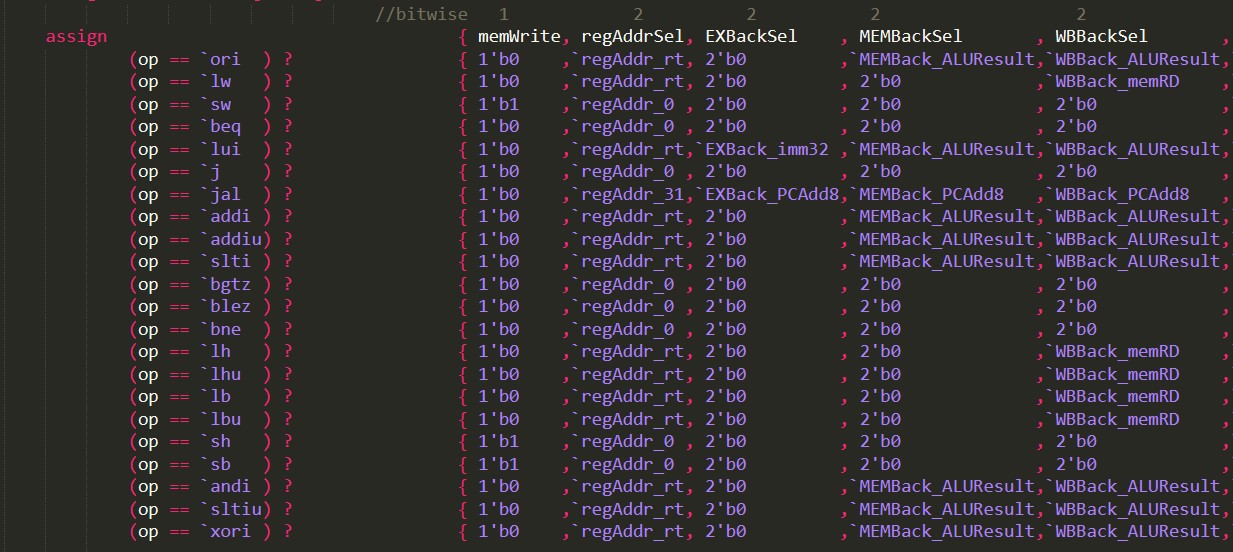
NPC模块内置了判定单元和计算单元来独立支持指令j、jal、jr 的跳转机制。

#### 2. 半字、字节存取

通过控制信号BECtrl和DECtrl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。

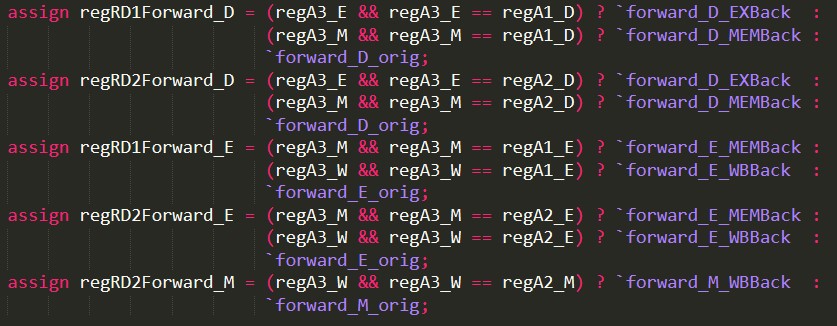
#### 3. 主控制器

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

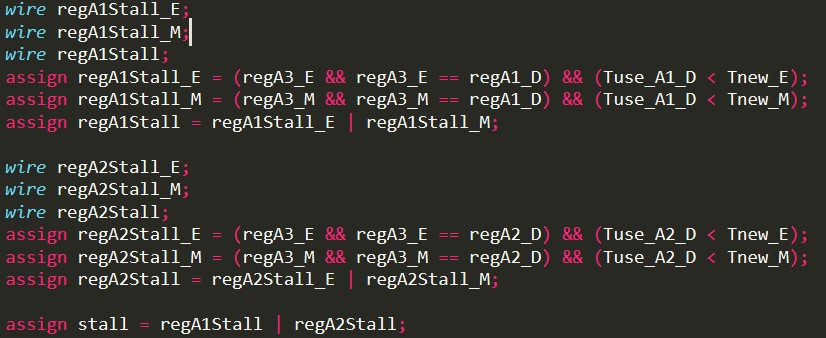
#### 4. 转发机制

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

#### 乘除槽

在MDU内部构建有限状态机，当start信号有效时，从初始状态变为等待状态，并保存操作数和控制信号。过5或10个周期后，写入HI和LO寄存器并返回初始状态。

### （四）异常、中断与外设读写

### 1. CP0（Coprocessor0）设计

模块定义：

module CP0 (

    input clk,

    input reset,

    input [4:0] A1,

    input [4:0] A2,

    input [31:0] WD,

    input [31:0] PC,

    input [4:0] excCode,

    input [5:0] HWInt,

    input WE,

    input EXLReset,

    input BD,

    output intExcReq,

    output reg [31:0] EPC,

    output [31:0] RD,

    output reg respTbInt

);

表23 CP0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 | A1 | I | 5 | CP0寄存器读地址 |
| 4 | A2 | I | 5 | CP0寄存器写地址 |
| 5 | WD | I | 32 | CP0寄存器写入数据 |
| 6 | PC | I | 32 | 宏观PC |
| 7 | excCode | I | 5 | 异常码 |
| 8 | HWInt | I | 6 | 外设中断 |
| 9 | WE | I | 1 | CP0寄存器写使能 |
| 10 | EXLReset | I | 1 | EXL复位信号 |
| 11 | BD | I | 1 | 延迟槽判断 |
| 12 | intExcReq | O | 1 | 中断异常响应信号 |
| 13 | EPC | O | 32 | EPC寄存器 |
| 14 | RD | O | 32 | CP0寄存器读出数据 |
| 15 | respTbInt | O | 1 | 响应testbench中断信号 |

内部逻辑说明：

内部含有SR、Cause、EPC、PrID四个寄存器，功能与SMRL一致。当写使能信号有效时，进行寄存器数据的写入。当异常码不为0且不在内核态（EXL！=1）时，响应异常；当全局中断使能有效、不在内核态、外部中断信号HTInt有效且相应位没有被屏蔽（(IM & HTInt) != 0）时，响应中断；当异常和中断需要同时响应时，中断优先级更高。进入handler时，将EXL设为1，宏观PC写入EPC，宏观BD写入Cause[31]，excCode写入Cause[6:2]，同时每周期更新IP。

### 2. Bridge与IO设计

模块定义：

module Bridge (

    input [31:0] addr,

    input [3:0] byteEn,

    input [31:0] DMRD,

    input [31:0] TC1RD,

    input [31:0] TC2RD,

    output [3:0] DMByteEn,

    output TC1Write,

    output TC2Write,

    output [31:0] RD

);

表24 Bridge端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | addr | I | 32 | 读取或写入地址 |
| 2 | byteEn | I | 4 | 字节使能 |
| 3 | DMRD | I | 32 | DM读取数据 |
| 4 | TC1RD | I | 32 | 计时器1读取数据 |
| 5 | TC2RD | I | 32 | 计时器2读取数据 |
| 6 | DMByteEn | O | 4 | DM字节使能 |
| 7 | TC1Write | O | 1 | TC1写使能 |
| 8 | TC2Write | O | 1 | TC2写使能 |
| 9 | RD | O | 32 | 读取数据 |

内部逻辑说明：

采用和高老板ppt相同的方式，通过地址范围来判断是命中DM、命中TC1还是命中TC2。如果命中DM，那么DM字节使能与原字节使能一致，否则为0，同时读出数据选择DM的数据；如果命中TC1，那么当原字节使能为4’b1111时，TC1的写使能信号有效，否则无效，同时读出数据选择TC1的数据；对于TC2同理。

## 二、测试方案

### （一）典型测试样例

1. testpoint0:测试除AdEL\_Instr之外的异常

.ktext 0x4180

mfc0 $k0, $12

mfc0 $k0, $13

mfc0 $k0, $14

addiu $k0, $k0, 4

mtc0 $k0, $14

eret

mfc0 $k0, $14

.text

ori $s0, $0, 0x1001

ori $s1, $0, 0x1

mtc0 $s0, $12

# AdEL\_DM

lw $2, 1($0)

lh $2, 1($0)

lhu $2, 1($0)

li $3, 2147483647

lw $2, 1($3)

lw $2, -1($0)

lw $2, 0x3000($0)

lw $2, 0x7f1c($0)

# AdEL\_Timer

lh $2, 0x7f00($0)

lhu $2, 0x7f00($0)

lb $2, 0x7f00($0)

mtc0 $s1, $12

lbu $2, 0x7f00($0)

lh $2, 0x7f10($0)

mtc0 $s0, $12

lhu $2, 0x7f10($0)

lb $2, 0x7f10($0)

lbu $2, 0x7f10($0)

# AdES\_DM

sw $2, 1($0)

sh $2, 1($0)

li $3, 2147483647

sw $2, 1($3)

sw $2, -1($0)

sw $2, 0x3000($0)

sw $2, 0x7f1c($0)

# AdES\_Timer

sw $2, 0x7f08($0)

sh $2, 0x7f00($0)

mtc0 $s1, $12

sb $2, 0x7f00($0)

sw $2, 0x7f18($0)

mtc0 $s0, $12

sh $2, 0x7f10($0)

sb $2, 0x7f10($0)

# RI

movz $0, $0, $0

# Ov

lui $1, 0x7fff

lui $2, 0x7fff

lui $3, 0xffff

add $4, $1, $2

addi $4, $1, 0x7fff

sub $4, $1, $3

end:

beq $0, $0, end

nop

1. testpoint1：测试AdEL\_Instr异常（跳转到错误地址引起）

.ktext 0x4180

mfc0 $k0, $12

mfc0 $k0, $13

mfc0 $k0, $14

addiu $k1, $k1, 8

mtc0 $k1, $14

eret

mfc0 $k0, $14

.text

ori $s0, $0, 0x1001

ori $s1, $0, 0x1

mtc0 $s0, $12

# AdEL\_Instr

ori $k1, $0, 0x301c

ori $1, $0, 0x300e

ori $2, $0, 0x2fff

ori $3, $0, 0x6ffd

jr $1

nop

mtc0 $s1, $12

jr $2

nop

mtc0 $s0, $12

jr $3

nop

jalr $31, $1

nop

jalr $31, $2

nop

jalr $31, $3

nop

end:

beq $0, $0, end

nop

1. Testpoint2:测试延迟槽相关异常

.ktext 0x4180

mfc0 $k0, $12

mfc0 $k0, $13

mfc0 $k0, $14

addiu $k0, $k0, 8

mtc0 $k0, $14

eret

mfc0 $k0, $14

.text

ori $s0, $0, 0x1001

ori $s1, $0, 0x1

mtc0 $s0, $12

# BranchDelay

ori $1, $0, 2

sw $1, 0($0)

lw $2, 0($0)

beq $2, $2, TAG1

lw $0, 1($0)

TAG1:

add $2, $1, $0

mtc0 $s1, $12

beqz $2, TAG2

sw $0, 1($0)

TAG2:

mtc0 $s0, $12

mult $1, $1

j TAG3

mflo $1

TAG3:

jal TAG4

movz $0, $0, $0

TAG4:

lui $1, 0x7fff

lui $2, 0x7fff

lui $3, 0xffff

bne $1, $2, TAG5

add $4, $1, $2

TAG5:

mtc0 $s1, $12

bgtz $0, TAG6

mtc0 $s0, $12

TAG6:

end:

beq $0, $0, end

nop

1. Testpoint3: 测试计时器模式0与IO

.text

ori $t0, $0, 0x1c01 # IE -> 1, IM -> 6'b000111

mtc0 $t0, $12

ori $t1, $0, 7 # preset -> 7

sw $t1, 0x7f04($0)

ori $t1, $0, 11

sw $t1, 0x7f14($0)

ori $t2, $0, 9 # ctrl -> 4'b1001

sw $t2, 0x7f00($0)

sw $t2, 0x7f10($0)

ori $t3, $0, 1 # ctrl -> 4'b0001

ori $t4, $0, 8 # ctrl -> 4'b1000

ori $s0, 100

for\_begin:

beq $s1, $s0, for\_end

sw $t3, 0x7f00($0)

sw $t3, 0x7f10($0)

lw $a1, 0x7f08($0)

mfc0 $a2, $13

sw $t2, 0x7f00($0)

sw $t2, 0x7f10($0)

addiu $s1, $s1, 1

j for\_begin

for\_end:

nop

nop

sw $t3, 0x7f00($0)

sw $t3, 0x7f10($0)

end:

beq $0, $0, end

nop

.ktext 0x4180

mfc0 $a2, $12

mfc0 $a2, $13

mfc0 $a2, $14

sw $t2, 0x7f00($0)

sw $t2, 0x7f10($0)

eret

1. Testpoint4: 测试计时器模式1与IO

.text

ori $t0, $0, 0x1c01 # IE -> 1, IM -> 6'b000111

mtc0 $t0, $12

ori $t1, $0, 7 # preset -> 7

sw $t1, 0x7f04($0)

ori $t1, $0, 11

sw $t1, 0x7f14($0)

ori $t2, $0, 11 # ctrl -> 4'b1011

sw $t2, 0x7f00($0)

sw $t2, 0x7f10($0)

ori $t3, $0, 1 # ctrl -> 4'b0011

ori $t4, $0, 8 # ctrl -> 4'b1010

ori $s0, 100

for\_begin:

beq $s1, $s0, for\_end

sw $t3, 0x7f00($0)

sw $t3, 0x7f10($0)

lw $a1, 0x7f08($0)

mfc0 $a2, $13

sw $t2, 0x7f00($0)

sw $t2, 0x7f10($0)

addiu $s1, $s1, 1

j for\_begin

for\_end:

nop

nop

sw $t3, 0x7f00($0)

sw $t3, 0x7f10($0)

end:

beq $0, $0, end

nop

.ktext 0x4180

mfc0 $a2, $12

mfc0 $a2, $13

mfc0 $a2, $14

sw $t2, 0x7f00($0)

sw $t2, 0x7f10($0)

eret

1. Testpoint5~n：自动生成的数据，见测试样例生成器

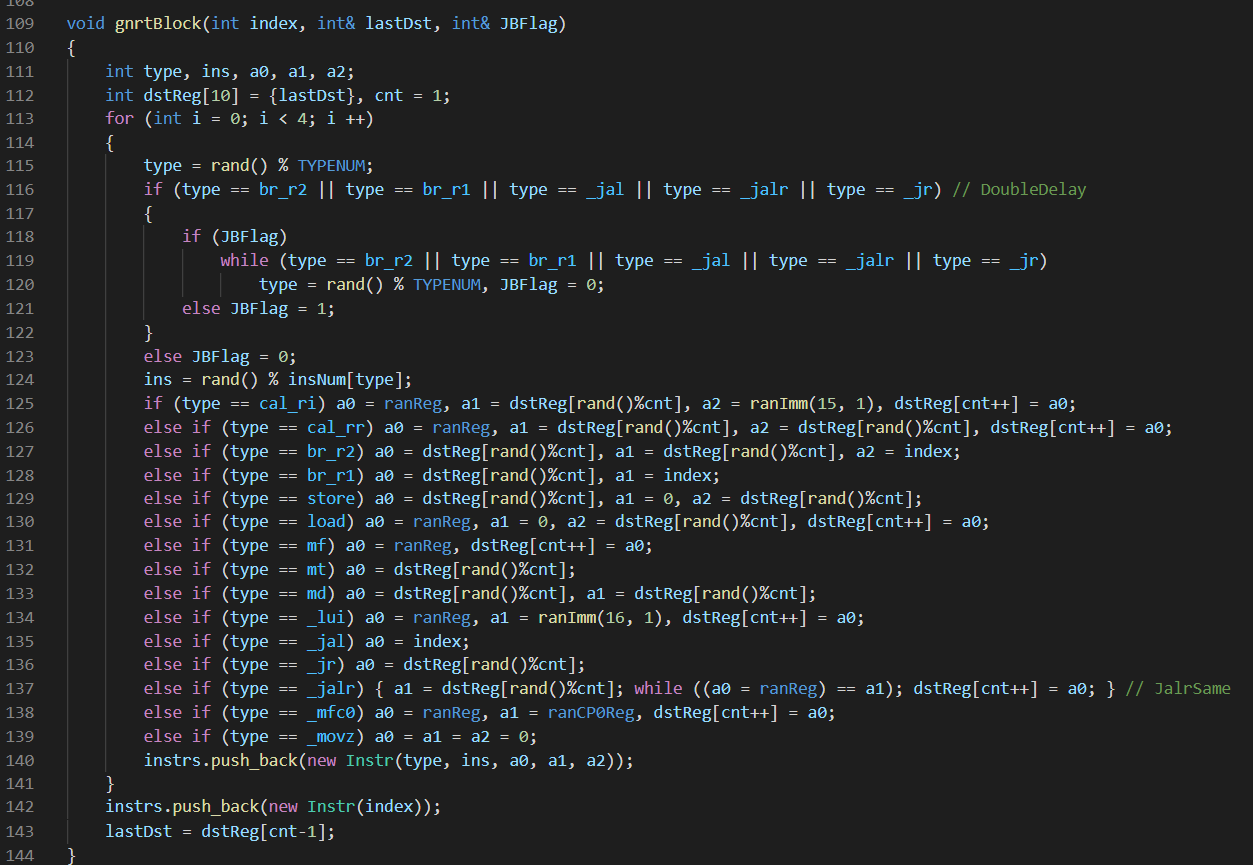
### （二）自动测试工具

#### 1. 测试样例生成器

运行环境：win10 g++ 11.1.0

程序大致思路为：

1. 利用ori初始化$28-$31寄存器。
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. 在生成代码块时只检查DoubleDelay、JalrSame，对于其他可能产生的异常，作为异常的测试点

图5 测试样例生成器部分代码

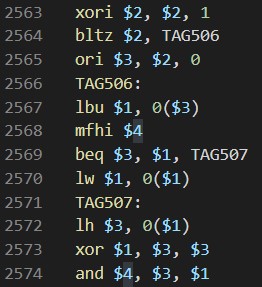


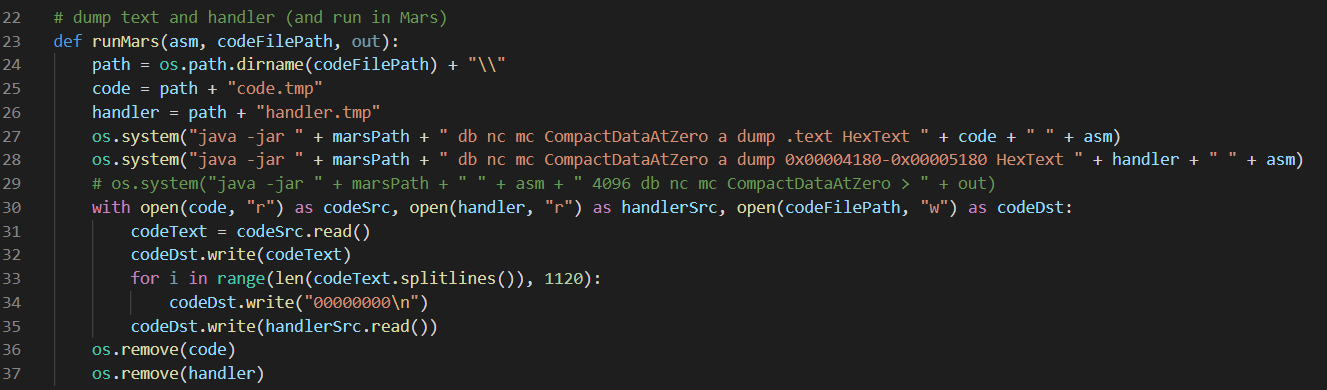
图6 生成的部分测试样例

#### 自动执行脚本

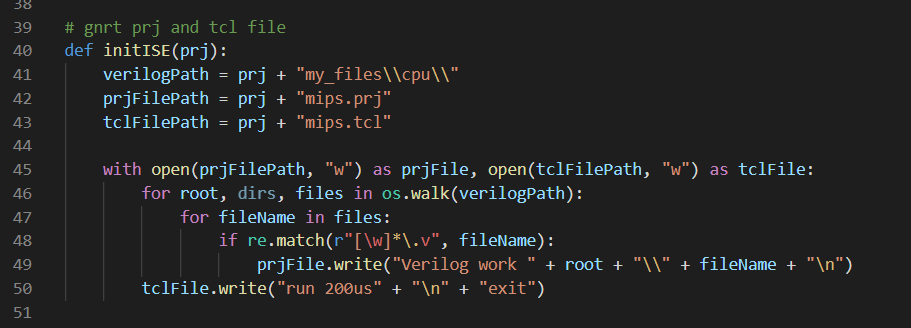
在P6基础上新增了对拍功能，以及全面测试外部中断

运行环境：win10 64位 python 3.9.6

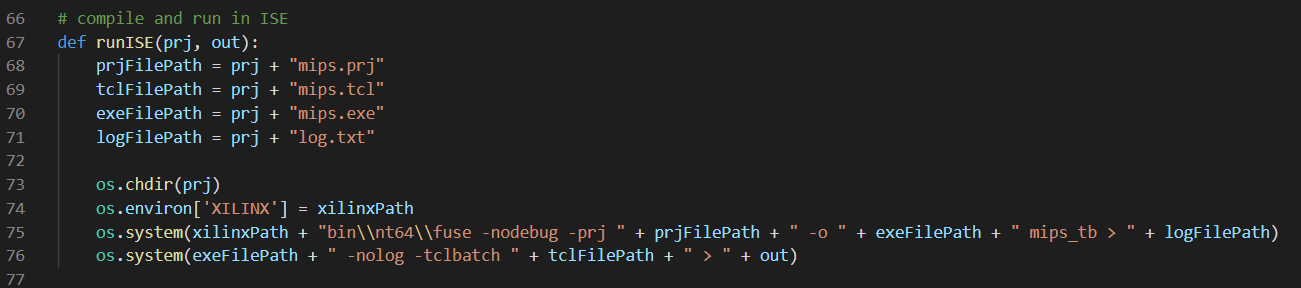
步骤1：导出code与handler



步骤2：生成prj和tcl文件



步骤3：分别编译运行两个CPU，进行仿真



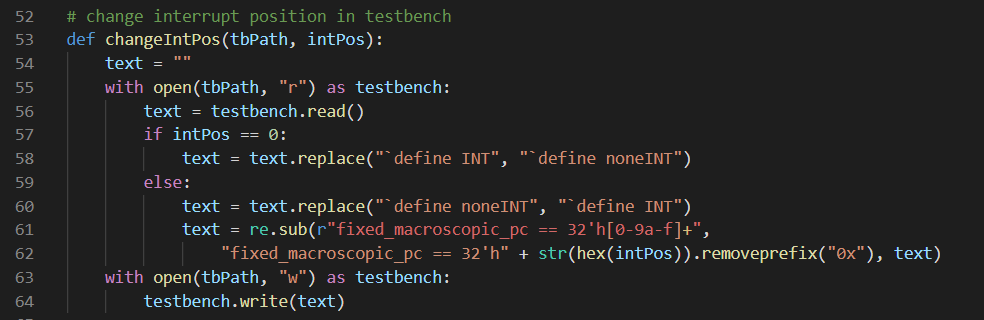
步骤4：将两个运行结果进行文本比对，如果出错则给出错误信息



### 测试模式：

有三种测试模式选择：无外部（testbench）中断，对某一个pc产生外部中断，对所有pc分别产生一次外部中断信号。

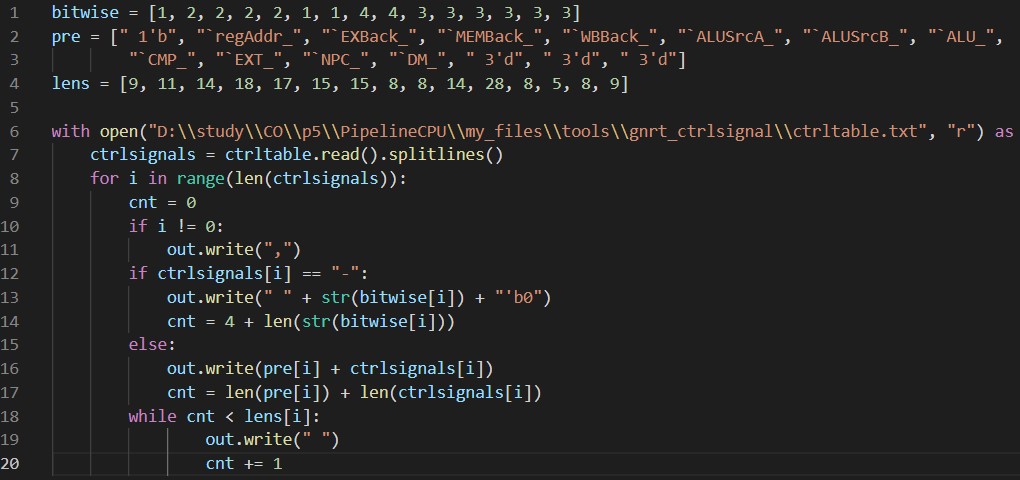
图为自动修改mips\_tb的部分



### （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。



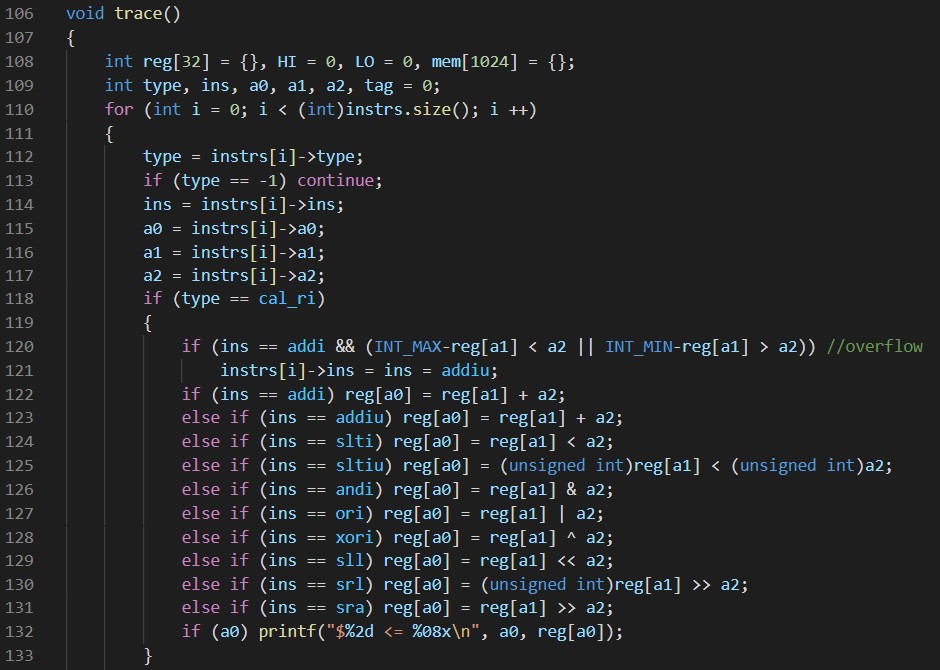
#### 2.自动生成模块端口：

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。



#### 模拟mips运行：

用c++模拟mips程序的运行，主要用于编写样例生成程序时debug



#### 自动分析数据：

通过python自带的zipfile库对机器码进行压缩，再放入官方提供的分析程序中进行分析。



## 三、外设控制与读写功能的实现（P8相关）

### （一）顶层模块定义

### 1. Clock

模块定义：

module Clock(

input         clk\_fpga,

output        clk\_cpu,

output        clk\_IMDM

 );

表1 Clock端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | Clk\_fpga | I | 1 | FPGA的时钟信号 |
| 2 | Clk\_cpu | O | 1 | CPU的时钟信号 |
| 3 | Clk\_IMDM | O | 1 | IM、DM的时钟信号 |

内部逻辑说明：

使用IP核生成，可将FPGA的25MHz时钟频率转换为CPU的60MHz时钟频率以及IM、DM的120MHz时钟频率。

### CPU（Central Processing Unit）

模块定义：

module CPU (

    input clk,

    input reset,

    input [5:0] HWInt,

    input [31:0] instr\_F,

    input [31:0] memRD,

    output [31:0] PC\_F,

    output [31:0] memAddr,

    output [31:0] memWD,

    output [3:0] byteEn,

    output [31:0] PC\_M,

    output regWrite,

    output [4:0] regAddr,

    output [31:0] regWD,

    output [31:0] PC\_W,

    output respTbInt

);

表2 CPU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 | 同步复位信号 |
| 3 | HWInt | I | 6 | 硬件中断 |
| 4 | Instr\_F | I | 32 | F级指令 |
| 5 | memRD | I | 32 | 内存读取数据 |
| 6 | PC\_F | O | 32 | F级PC |
| 7 | memAddr | O | 32 | 内存写入地址 |
| 8 | memWD | O | 32 | 内存写入数据 |
| 9 | bytenEn | O | 4 | 字节使能 |
| 10 | PC\_M | O | 32 | M级PC |
| 11 | regWrite | O | 1 | GRF写使能 |
| 12 | regAddr | O | 5 | GRF写入地址 |
| 13 | regWD | O | 32 | GRF写入数据 |
| 14 | PC\_W | O | 32 | W级PC |
| 15 | respTbInt | O | 1 | 响应tb中断信号 |

内部逻辑说明：

内部包含了控制器和数据通路，实现了控制器和数据通路间、CPU和外部的数据与信号的交互

### 3. IM（Instruction Memory）

模块定义：

module IM (

input clka,

input [11:0] addra,

output [31:0] douta

 );

表3 IM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clka | I | 1 | 时钟信号 |
| 2 | addra | I | 12 | 12位地址 |
| 3 | douta | O | 32 | 读取数据 |

内部逻辑说明：

使用IP核生成，时钟频率经过调制，为120MHz，内存大小为4096 Words，需要等待一周期才能获得有效数据。

### 4. Bridge

模块定义：

module Bridge (

    input [31:0] addr,

    input [3:0] byteEn,

    input [31:0] DMRD,

    input [31:0] TCRD,

    input [31:0] DTRD,

    input [31:0] UARTRD,

    input [31:0] IORD,

    output [3:0] DMByteEn,

    output TCWrite,

    output [3:0] DTByteEn,

    output UARTWrite,

    output [3:0] IOByteEn,

    output [31:0] RD

);

表4 Bridge端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | addr | I | 32 | 写入地址 |
| 2 | byteEn | I | 4 | 原字节使能 |
| 3 | DMRD | I | 32 | DM读取数据 |
| 4 | TCRD | I | 32 | 计时器读取数据 |
| 5 | DTRD | I | 32 | 数码管读取数据 |
| 6 | UARTRD | I | 32 | 串口读取数据 |
| 7 | IORD | I | 32 | 通用IO读取数据 |
| 8 | DMByteEn | O | 4 | DM字节使能 |
| 9 | TCWrite | O | 1 | 计时器写使能 |
| 10 | DTByteEn | O | 4 | 数码管字节使能 |
| 11 | UARTWrite | O | 1 | 串口写使能 |
| 12 | IOByteEn | O | 4 | 通用IO字节使能 |
| 13 | RD | O | 32 | 读取数据 |

内部逻辑说明：

通过地址范围来判断是命中DM、命中TC还是别的外设。以DM为例，如果命中DM，那么DM字节使能与原字节使能一致，否则为0，同时读出数据选择DM的数据。

### 5. DM（Data Memory）

模块定义：

module Clock (

input clka,

Input [3:0] wea,

input [11:0] addra,

Input [31:0] dina,

output [31:0] douta

 );

表5 DM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clka | I | 1 | 时钟信号 |
| 2 | wea | I | 4 | 字节使能 |
| 3 | addra | I | 12 | 12位地址 |
| 4 | dina | I | 32 | 写入数据 |
| 5 | douta | O | 32 | 读取数据 |

内部逻辑说明：

使用IP核生成，时钟频率经过调制，为120MHz，内存大小为3072 Words，需要等待一周期才能获得有效数据。

### 6. TC（Timer Counter）

模块定义：

module TC(

input clk,

input reset,

input [3:2] Addr,

input WE,

input [31:0] Din,

output [31:0] Dout,

output IRQ

);

表6 TC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 | 同步复位信号 |
| 3 | Addr | I | 2 | 地址信号 |
| 4 | WE | I | 1 | 写使能信号 |
| 5 | Din | I | 32 | 写入数据 |
| 6 | Dout | O | 32 | 读取数据 |
| 7 | IRQ | O | 1 | 中断信号 |

内部逻辑说明：

与p7计时器无差别，见计时器说明文档

### 7. DT（Digital Tube）

模块定义：

module DT (

    input clk,

    input reset,

    input [3:0] byteEn,

    input Addr,

    input [31:0] WD\_orig,

    output [31:0] RD,

    output [7:0] code0,

    output [7:0] code1,

    output [7:0] code2,

    output reg [3:0] sel0,

    output reg [3:0] sel1,

    output reg sel2

);

表7 DT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 | 同步复位信号 |
| 3 | byteEn | I | 4 | 字节使能 |
| 4 | Addr | I | 1 | 地址信号 |
| 5 | WD\_orig | I | 32 | 原始写入数据 |
| 6 | RD | O | 32 | 读取数据 |
| 7 | Code0 | O | 8 | 0-3位数码管编码 |
| 8 | Code1 | O | 8 | 4-7位数码管编码 |
| 9 | Code2 | O | 8 | 第8位数码管编码 |
| 10 | Sel0 | O | 4 | 0-3位数码管选择信号 |
| 11 | Sel1 | O | 4 | 4-7位数码管选择信号 |
| 12 | Sel2 | O | 1 | 第8位数码管选择信号 |

内部逻辑说明：

0-3号和4-7号数码管分别为两个四位共阳极八段数码管，采用不断切换选择信号的方式进行动态显示。内部寄存器定义与官方给的相同，支持字节读写。

### 8. MiniUART（Mini Universal Asynchronous Receiver/Transmitter）

模块定义：

module MiniUART(

Input CLK\_I,

Input [31:0] DAT\_I,

Output [31:0] DAT\_O,

Input RST\_I,

Input [4:2] ADD\_I,

Input STB\_I,

Input WE\_I,

Output ACK\_O,

Input RxD,

Output TxD,

Output reg interrupt,

Input respInt

);

表8 UART端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | CLK\_I | I | 1 | 时钟信号 |
| 2 | DAT\_I | I | 32 | 写入数据 |
| 3 | DAT\_O | O | 32 | 读取数据 |
| 4 | RST\_I | I | 1 | 同步复位信号 |
| 5 | ADD\_I | I | 3 | 地址信号 |
| 6 | STB\_I | I | 1 | 选通信号 |
| 7 | WE\_I | I | 1 | 写使能信号 |
| 8 | ACK\_O | O | 1 | 操作结束方式信号 |
| 9 | RxD | I | 1 | 接收数据 |
| 10 | TxD | O | 1 | 发送数据 |
| 11 | interrupt | O | 1 | 中断信号 |
| 12 | respInt | I | 1 | 中断响应信号 |

内部逻辑说明：

在官方的基础上进行了如下修改：把时钟频率调整到60MHz（与自己的cpu一致），并且加入中断功能，当读取到完整有效数据时，中断信号持续输出1，直到接收到来自CPU的响应信号后停止。

### 9. IO（General-Purpose Input Output）

模块定义：

module IO (

    input clk,

    input reset,

    input [3:0] byteEn,

    input [4:2] Addr,

    input [31:0] WD\_orig,

    input [31:0] dips0\_3,

    input [31:0] dips4\_7,

    input [7:0] key,

    output [31:0] RD,

    output [31:0] LED

);

表9 IO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 | 同步复位信号 |
| 3 | byteEn | I | 4 | 字节使能 |
| 4 | Addr | I | 3 | 地址信号 |
| 5 | WD\_orig | I | 32 | 原始写入数据 |
| 6 | dips0\_3 | I | 32 | 0-3组拨码开关 |
| 7 | dips4\_7 | I | 32 | 4-7组拨码开关 |
| 8 | key | I | 8 | 按键开关 |
| 9 | RD | O | 32 | 读取数据 |
| 10 | LED | O | 32 | LED控制信号 |

内部逻辑说明：

控制0-8组共128位拨码开关、8个按键开关以及32个LED灯。内部寄存器定义与官方一致，支持字节存取。

### 10.其他IP核（以有符号乘法器Mult为例）

模块定义：

module Mult (

Input clk,

Input [31:0] a,

Input [31:0] b,

Output [63:0] p

);

表10 Mult定义

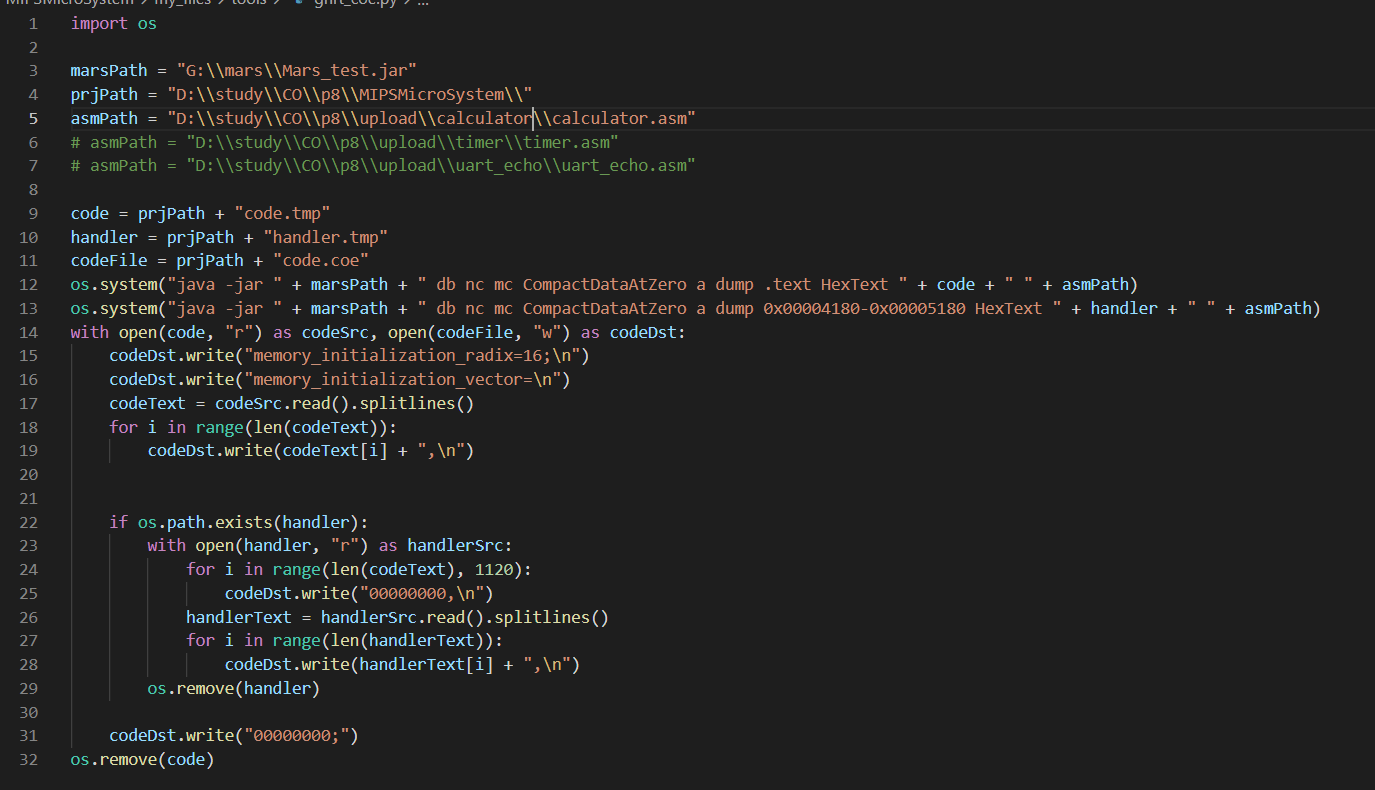
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | a | I | 32 | 操作数a |
| 3 | b | I | 32 | 操作数b |
| 4 | p | O | 64 | 乘法结果 |

内部逻辑说明：

内部采用流水线结构，需要经过五个周期得到结果。分别用有符号乘法器Mult，无符号乘法器Multu，有符号除法器Div，无符号除法器Divu代替原来MDU中乘除法运算，便可支持MDU相关指令。

### （二）自动化工具

自动导出指令代码并生成coe文件：



### 汇编程序

### 简易计算器

功能：支持2个32位操作数的8中运算，0-7号按键开关分别对应与、或、异或、加、减、左移、逻辑右移、算术右移，当没有按键按下时，保持上次结果，当多个按键按下时，靠右的按键优先级最高。

代码思路：无限循环并检测操作数是否发生改变，当发生改变且操作有效时，分别进行不同的运算并输出结果到数码管。选择不同的运算采用了跳转表的结构（类似switch-case语句），根据最右侧按键的位置计算出跳转地址，然后过去进行计算，再跳转到同一的结束地址。

代码：

# 0->7 : & | ^ + - << >> >>>

# priority 0 > ... > 7

# li $t0, 1

# li $t1, 2

# li $t2, 0x8

# sw $t0, 0x7f54($0)

# sw $t1, 0x7f50($0)

# sb $t2, 0x7f58($0)

endless\_loop:

    # get operands and operator

    lw $a0, 0x7f54($0)

    lw $a1, 0x7f50($0)

    lb $a2, 0x7f58($0)

    # jump table

    li $t0, 0

    while\_1\_begin:

    beqz $a2, while\_1\_end

    nop

        and $t1, $a2, 1

        beqz $t1, if\_1\_end

        nop

            la $ra, op\_begin

            addu $ra, $ra, $t0

            jr $ra

            nop

        if\_1\_end:

        srl $a2, $a2, 1

        addiu $t0, $t0, 8

    j while\_1\_begin

    nop

    while\_1\_end:

    j endless\_loop

    nop

op\_begin:

    j op\_end

    and $s0, $a0, $a1

    j op\_end

    or $s0, $a0, $a1

    j op\_end

    xor $s0, $a0, $a1

    j op\_end

    addu $s0, $a0, $a1

    j op\_end

    subu $s0, $a0, $a1

    j op\_end

    sllv $s0, $a0, $a1

    j op\_end

    srlv $s0, $a0, $a1

    j op\_end

    srav $s0, $a0, $a1

op\_end:

    sw $s0, 0x7f40($0)

    j endless\_loop

    nop

### 计时器

功能：从0-3组拨码开关读取初始值，随后开始倒计时，倒计时到0时，32个LED灯亮一秒，随后重新开始计时。当复位或初始值改变时，也重新开始计时

代码思路：无限循环检测初始值是否发生改变，当改变时重新载入初始值进行计时并重置内部计时器。微系统内部的计时器作用是计算一秒的时间，将TC的preset设置为60000000（CPU的主频），然后使用模式0开始计时。每当TC倒计时结束并产生中断信号时，表示过去了1秒，进入中断处理程序以后，将显示在数码管上的数值减1，然后将TC的使能信号ctrl[0]置1，重新开始计时。

代码：

.text

    # li $t0, 10

    # sw $t0, 0x7f50($0)

    # set default values and init

    li $t0, 0x1401      # 1\_0100\_0000\_0001

    mtc0 $t0, $12

    li $t0, 60000000    # frequency -> preset

    sw $t0, 0x7f04($0)

    li $a0, 9  # 1001

    li $a1, -1 # 111...111

    li $s0, 0 # cnt

    li $s1, 0 # pre-value

    loop:

        lw $t0, 0x7f50($0)

        beq $s1, $t0, if\_1\_end

        nop

            sw $0, 0x7f00($0) # stop count

            addu $s1, $0, $t0

            addu $s0, $0, $t0 # reload

            sw $0, 0x7f60($0) # LED off

            sw $a0, 0x7f00($0) # restart count

            if\_2\_end:

        if\_1\_end:

        sw $s0, 0x7f40($0) # digitalTube

    j loop

    nop

.ktext 0x4180

    sw $a0, 0x7f00($0)

    sw $0, 0x7f60($0) # LED off

    bnez $s0, if\_3\_end

    nop

        addu $s0, $0, $s1 # reload

        eret

    if\_3\_end:

    addiu $s0, $s0, -1

    bnez $s0, if\_4\_end

    nop

        sw $a1, 0x7f60($0) # LED on

        eret

    if\_4\_end:

    eret

### UART回显

功能：将串口输入直接原样输出，并且在数码管上显示接收的字符数，支持9600,19200,38400，57600，115200五种波特率模式，通过0-3号用户按键来控制，当0个或>1个按键按下时，使用9600模特率，最右侧按键按下时，采用19200波特率，以此类推。

代码思路：当MiniUART模块成功读取一个完整有效数据时会产生中断信号，随后进入中断处理程序，先检测LSR寄存器，如果接受数据有效，就将该数据读出并存入缓冲区，如果发送保持器空，就从缓冲区取一个字符并发送。

由于接收数据的采样时间远远大于发送数据的时间，所以每当数据读取好时，发送保持器总是为空，因此暂时可以不设置缓冲区。当接收数据的时间与发送数据时间区别不大时，存在缓冲区无法及时输出的情况，因此本程序也可加入轮询功能（目前为不影响性能已注释掉），方法是将内部计时器设置为模式1，每隔一定周期进行一次中断，检测缓冲区是否有未发送的数据。

代码：

.text

li $sp, 0x2fff  # buffer

li $t0, 0x1001

mtc0 $t0, $12

li $t0, 0xb

sb $t0, 0x7f44($0)

li $t0, 0x96

sb $t0, 0x7f41($0)

# li $t0, 1000        # check per 1000 cycles

# sw $t0, 0x7f04($0)

# li $t0, 11          # 1011

# sw $t0, 0x7f00($0)  # start timer in mode 1

li $s0, 0

loop:

    lb $a0, 0x7f58($0)

    andi $a0, $a0, 0xf

    beq $s0, $a0, loop

    nop

    addiu $s0, $a0, 0

    bne $a0, 1, if\_3\_else\_1

    nop

        li $t0, 389

        li $t1, 3124

        li $t2, 0x92

        li $t3, 0x01

        j if\_3\_end

        nop

    if\_3\_else\_1:

    bne $a0, 2, if\_3\_else\_2

    nop

        li $t0, 194

        li $t1, 1561

        li $t2, 0x84

        li $t3, 0x03

        j if\_3\_end

        nop

    if\_3\_else\_2:

    bne $a0, 4, if\_3\_else\_3

    nop

        li $t0, 129

        li $t1, 1040

        li $t2, 0x76

        li $t3, 0x05

        j if\_3\_end

        nop

    if\_3\_else\_3:

    bne $a0, 8, if\_3\_else\_4

    nop

        li $t0, 64

        li $t1, 519

        li $t2, 0x52

        li $t3, 0x11

        j if\_3\_end

        nop

    if\_3\_else\_4:

    li $t0, 780

    li $t1, 6249

    li $t2, 0x96

    li $t3, 0x00

    if\_3\_end:

    sw $t0, 0x7f34($0)

    sw $t1, 0x7f38($0)

    sb $t2, 0x7f41($0)

    sb $t3, 0x7f42($0)

beq $0, $0, loop

nop

.ktext 0x4180

lw $t0, 0x7f30($0)

andi $t1, $t0, 1

andi $t2, $t0, 32

beqz $t1, if\_1\_end

nop

    lw $t0, 0x7f20($0)

    sb $t0, 0($sp)

    addiu $sp, $sp, -1

if\_1\_end:

beqz $t2, if\_2\_end

nop

bge $sp, 0x2fff, if\_2\_end

nop

    addiu $sp, $sp, 1

    lb $t0, 0($sp)

    sw $t0, 0x7f20($0) # echo

    lb $t0, 0x7f43($0)

    addiu $t0, $t0, 1

    sb $t0, 0x7f43($0) # update digital tube

if\_2\_end:

eret