计算机组成原理P6实验报告

彭杰奇15061169

1. 数据通路设计

F级功能部件：

1.IFU模块

（1）基本描述

IFU主要功能是完成取指令功能。IFU内部包括了PC、IM(指令存储器)以及其他相关逻辑。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IFU.v | IFU(nextPC, Clk, Reset, PC\_En, PC4, Instr);  input [31:0] nextPC;  input Clk;  input Reset;  input PC\_En;  output [31:0] Instr;  output [31:0] PC4; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| nextPC[31:0] | I | 输入PC的下一条指令地址 |
| Reset | I | 复位信号，1:有效，0:无效 |
| Clk | I | 时钟信号 |
| PC\_En | I | IFU内部PC的使能端，1:有效，0:无效 |
| Instr[31:0] | O | 当前指令输出 |
| PC4[31:0] | O | 当前PC下PC + 4的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，PC设置为0x00003000 |
| 2 | 取指令 | 根据PC当前值从IM中取指令输出 |
| 3 | 输出PC+4 | PC4 为当前PC下 PC + 4 的值 |

2.IM模块

（1）基本描述

IM是指令存储模块，由一个32bit×2048字的存储器组成，其功能是保存指令，并根据输入的PC输出相应指令。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IM.v | IM(Addr,Instr);  input [11:2] Addr; // 输入的指令地址  output [31:0] Instr; // 输出的指令 |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[11:2] | I | 输入的指令地址 |
| Instr[31:0] | O | 输出的指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出指令 | Instr 🡨 im[Addr] |

IF\_ID流水线寄存器：

(1)模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IF\_ID.v | IF\_ID(IM, ADD4, Clk, Reset, IR\_D\_En, IR\_D, PC4\_D);  input [31:0] IM;  input [31:0] ADD4;  input Clk;  input Reset;  input IR\_D\_En;  output [31:0] IR\_D;  output [31:0] PC4\_D; |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 清空寄存器 |
| 2 | 存数取数 | 将前一级的值存入寄存器中，将后一级的值输出 |
| 3 | 冻结 | 冻结IR\_D |

D级功能部件：

1.GRF模块

（1）基本描述

GRF模块为通用寄存器堆，主要由32个具有写使能端的32位寄存器组成，能够同时根据由rs和rt输入的地址从其中两个寄存器中读出数据，并根据wr中输入的地址向其中一个寄存器写入数据。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| GRF.v | GRF(rs, rt, wr, WData, Clk, Reset, RegWrite, RData1, RData2);  input [4:0] rs;  input [4:0] rt;  input [4:0] wr;  input [31:0] WData;  input Clk;  input Reset;  input RegWrite;  output [31:0] RData1;  output [31:0] RData2; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rs[4:0] | I | rs寄存器的地址 |
| rt[4:0] | I | rt寄存器的地址 |
| wr[4:0] | I | 要写入的寄存器的地址 |
| WData[31:0] | I | 要写入的数据 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| RegWrite | I | 一般写使能信号，1:有效，0:无效 |
| RData1[31:0] | O | rs寄存器的值 |
| RData2[31:0] | O | rt寄存器的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | RData1🡨(GRF[rs])  RData2🡨(GRF[rt]) |
| 2 | 写数据 | RegWrite有效时，(GPR[wr])🡨WData |
| 3 | 清零 | 复位信号有效时，GRF中所有寄存器都清零 |

2.EXT模块

（1）基本描述

EXT模块的作用是将16位立即数扩展为32位。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EXT.v | EXT(Imm\_16, ExtOp, Imm\_32);  input [15:0] Imm\_16;  input [1:0] ExtOp;  output [31:0] Imm\_32; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm\_16[15:0] | I | 要扩展的16位立即数 |
| ExtOp[1:0] | I | 扩展方式选择信号  2’b00:符号扩展  2’b01:后接16位0  2’b10:无符号扩展 |
| Imm\_32[31:0] | O | 扩展后的32位立即数 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 位数扩展 | ExtOp为2’b00时，16位立即数正常符号扩展为32位  ExtOp为2’b01时，16为立即数后接16位0扩展为32位  ExtOp为2’b10时，16为立即数无符号扩展为32位 |

3.CMP模块

（1）基本描述

CMP模块用来比较输入的两个数据是否相等以及数据和0的比较，用于beq指令是否跳转和其他b类型指令跳转的判断

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| CMP.v | CMP(A1, A2, equal);  input [31:0] A1;  input [31:0] A2;  output equal;  output bgez;  output bgtz;  output blez;  output bltz; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1[31:0] | I | 第一个数据 |
| A2[31:0] | I | 第二个数据 |
| Equal | O | 输入的数据是否相等  1:相等  0:不相等 |
| bgez | O | A1大于等于0则为1，否则为0 |
| bgtz | O | A1大于0则为1，否则为0 |
| blez | O | A1小于等于0则为1，否则为0 |
| bltz | O | A1小于0则为1，否则为0 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 等于判断 | equal 🡨 (A1==A2)?1:0 |

4.NPC模块

（1）基本描述

NPC模块能够输出当前指令为beq或J类型等跳转指令时下一条指令地址

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| NPC.v | NPC(PC4, Instr, J\_Sel, Branch, Zero, nPC);  input [31:0] PC4;  input [31:0] Instr;  input [1:0] J\_Sel;  input Branch;  input Zero;  output [31:0] nPC; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC4[31:0] | I | 来自PC4\_D寄存器 |
| Instr[31:0] | I | 来自IR\_D寄存器 |
| J\_Sel[1:0] | I | 当下J类型指令具体为:  2’b00:不是j类型指令  2’b01:指令为J  2’b10:指令为Jal  2’b11:指令为Jr |
| Branch | I | 当下指令是否为beq  1:是，0:不是 |
| Zero | I | 若为beq指令，比较的两个数据是否相等  1:相等，0:不相等 |
| nPC[31:0] | O | 下一条指令地址  J\_Sel为2’b01或2’b10:  nPC 🡨 PC[31:28]||index||0^2  Branch为1且Zero为1:  nPC 🡨 PC + 4 + Imm\_32||0^2  Branch 为1而Zero为0:  nPC🡨 PC4 + 4  其中:  PC = PC4-4 Index = Instr[25:0] Imm16 = Instr[15:0] |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出下一条PC地址 | nPC输出当指令为beq或者J类型指令时下一条指令的地址 |

ID\_EX流水线寄存器：

(1)模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ID\_EX.v | ID\_EX(IR\_D, PC4\_D, RD1, RD2, EXT, Clk, Reset, IR\_E\_Clr, IR\_E, PC4\_E, RS\_E, RT\_E, EXT\_E);  input [31:0] IR\_D;  input [31:0] PC4\_D;  input [31:0] RD1;  input [31:0] RD2;  input [31:0] EXT;  input Clk;  input Reset;  input IR\_E\_Clr;  output [31:0] IR\_E;  output [31:0] PC4\_E;  output [31:0] RS\_E;  output [31:0] RT\_E;  output [31:0] EXT\_E; |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 清空寄存器 |
| 2 | 存数取数 | 将前一级的值存入寄存器中，将后一级的值输出 |
| 3 | 清除 | 清除IR\_E的值 |

EX级功能部件：

1.ALU模块

（1）基本描述

ALU为算数逻辑单元，可以对输入的两个数据进行加、减、按位与和按位或操作，并能够判断输入数据是否相等。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ALU.v | ALU(A1, A2, ALUCtr, ALUResult);  input [31:0] A1;  input [31:0] A2;  input [2:0] ALUCtr;  output [31:0] ALUResult; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1[31:0] | I | 第一个运算数 |
| A2[31:0] | I | 第二个运算数 |
| ALUCtr[2:0] | I | ALU控制信号  2’b000:加法运算  2’b001:减法运算  2’b010:按位与运算  2’b011:按位或运算 |
| ALUResult[31:0] | O | ALU运算结果 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法运算 | ALUResult 🡨 A1+A2 |
| 2 | 减法运算 | ALUResult 🡨 A1-A2 |
| 3 | 按位与运算 | ALUResult 🡨 A1&A2 |
| 4 | 按位或运算 | ALUResult 🡨 A1|A2 |

2.Mult\_Div模块

（1）基本描述

乘除法部件支持 mult、multu、div、divu、mfhi、mflo、mthi 及 mtlo 这些乘除法相关指令，内置Hi和Lo寄存器。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| Mult\_Div.v | Mult\_Div(A1, A2, MDOp, MTOp, Clk, Reset, start, busy, Hi, Lo);  input [31:0] A1;  input [31:0] A2;  input [2:0] MDOp;  input [1:0] MTOp;  input Clk;  input Reset;  input start;  output busy;  output [31:0] Hi;  output [31:0] Lo; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1[31:0] | I | 第一个运算数 |
| A2[31:0] | I | 第二个运算数 |
| MDOp[2:0] | I | 乘除法控制信号  001：有符号乘  010：无符号乘  011：有符号除  100：无符号除 |
| MTOp[1:0] | I | 写Hi与Lo寄存器控制信号  01：写Hi寄存器  10：写Lo寄存器 |
| start | I | 乘除法运算开始信号 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| busy | O | 乘除法延迟信号 |
| Hi | O | Hi寄存器输出 |
| Lo | O | Lo寄存器输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 乘法运算 | {Hi,Lo}🡨 A1\*A2 |
| 2 | 除法运算 | Lo 🡨 A1/A2 Hi 🡨 A1%A2 |

EX\_MEM流水线寄存器：

(1)模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EX\_MEM.v | EX\_MEM(IR\_E, PC4\_E, AO, RT\_E, Clk, Reset, IR\_M, PC4\_M, AO\_M, RT\_M);  input [31:0] IR\_E;  input [31:0] PC4\_E;  input [31:0] AO;  input [31:0] RT\_E;  input Clk;  input Reset;  output [31:0] IR\_M;  output [31:0] PC4\_M;  output [31:0] AO\_M;  output [31:0] RT\_M; |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 清空寄存器 |
| 2 | 存数取数 | 将前一级的值存入寄存器中，将后一级的值输出 |

MEM级功能部件：

1.DM模块

（1）基本描述

DM模块为数据存储器，由一个32bit \* 2048字的存储器构成，起始地址为0x00000000用于存储数据。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| DM.v | DM(Addr,Din,MemWrite,MemRead,Clk,Reset,Dout);  input [31:0] Addr;  input [31:0] Din;  input MemWrite;  input MemRead;  input [3:0] BE;  input Clk;  input Reset;  output [31:0] Dout; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[31:0] | I | 读/写DM的地址 |
| Din[31:0] | I | 要写入DM的数据 |
| MemWrite | I | 写DM的控制信号 |
| MemRead | I | 读DM的控制信号 |
| BE | I | 字节使能 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| Dout[31:0] | O | 从DM读出的数据 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 当MemRead为1时，ReadData 🡨 RAM(Addr[11:2]) |
| 2 | 写数据 | 当MemWrite为1时，RAM(Addr) 🡨 WriteData |
| 3 | 清零 | 复位信号有效时，存储器清零 |

MEM\_WB流水线寄存器：

(1)模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MEM\_WB.v | MEM\_WB(IR\_M, PC4\_M, AO\_M, DR\_M, Clk, Reset, IR\_W, PC4\_W, AO\_W, DR\_W);  input [31:0] IR\_M;  input [31:0] PC4\_M;  input [31:0] AO\_M;  input [31:0] DR\_M;  input Clk;  input Reset;  output [31:0] IR\_W;  output [31:0] PC4\_W;  output [31:0] AO\_W;  output [31:0] DR\_W; |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 清空寄存器 |
| 2 | 存数取数 | 将前一级的值存入寄存器中，将后一级的值输出 |

W级功能部件：

1.Load\_Ext模块

（1）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| Load\_Ext.v | Load\_Ext(Din,Addr,Op,Dout);  input [31:0] Din;  input [31:0] Addr;  input [2:0] Op;  output [31:0] Dout; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[31:0] | I | 原本读/写DM的地址 |
| Din[31:0] | I | 输入32位数据 |
| Op[2:0] | I | 数据扩展控制码  000: 无扩展  001: 无符号字节数据扩展  010: 有符号字节数据扩展  011: 无符号半字数据扩展  100: 有符号半字数据扩展 |
| Dout[31:0] | O | 扩展后的32位数据 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据扩展 | 根据数据扩展控制码对输入的数据进行扩展 |

1. 控制器设计
2. Controller模块定义

（1）基本描述

Controller模块为CPU控制器，可以根据输入指令的opcode和funct值输出各种控制信号。

（2）模块接口

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| Controller.v | Controller(Instr, RegDst, ALUSrc, MemtoReg, RegWrite, MemWrite, MemRead, ExtOp, Branch, J\_Sel, ALUCtr, PC\_Sel);  input [31:0] Instr;  output [1:0] RegDst;  output ALUSrc;  output [1:0] MemtoReg;  output RegWrite;  output MemWrite;  output MemRead;  output [1:0] ExtOp;  output Branch;  output [1:0] J\_Sel;  output [1:0] PC\_Sel;  output [2:0] ALUCtr; |

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Instr[31:0] | I | 指令 |
| RegDst[1:0] | O | 寄存器写入端地址控制  2’b00:选择rt字段  2’b01:选择rd字段  2’b10:选择31号寄存器 |
| ALUSrc | O | ALU输入端A2选择  0:选择MSRTE  1:选择EXT\_E |
| MemtoReg[1:0] | O | 寄存器堆写入端WD选择  2’b00:来自ALU输出  2’b01:来自DM输出  2’b10:来自PC4\_W+4 |
| RegWrite | O | 写寄存器控制信号 |
| MemWrite | O | 写DM控制信号 |
| MemRead | O | 读DM控制信号 |
| ExtOp[1:0] | O | EXT扩展方式控制信号 |
| Branch | O | 判断是否为beq指令 |
| J\_Sel[1:0] | O | 2’b00:其他指令  2’b01:J指令  2’b10:Jal指令  2’b11:Jr指令 |
| PC\_Sel[1:0] | O | 2’b00:IFU的nextPC选择PC+4  2’b01:IFU的nextPC选择nPC  2’b10:IFU的nextPC选择RData1 |
| ALUCtr[2:0] | O | ALU控制信号 |

1. Controller真值表

|  |  |  |  |
| --- | --- | --- | --- |
| Instr | Subu | addu | Jr |
| opcode | 000000 | 000000 | 000000 |
| funct | 100011 | 100001 | 001000 |
| RegDst[1:0] | 2’b01 | 2’b01 | 2’b01 |
| ALUSrc | 0 | 0 | 0 |
| MemtoReg[1:0] | 2’b00 | 2’b00 | 2’b00 |
| RegWrite | 1 | 1 | 0 |
| Branch | 0 | 0 | 0 |
| J\_Sel[1:0] | 2’b00 | 2’b00 | 2’b11 |
| ExtOp[1:0] | X | X | X |
| MemRead | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 |
| PC\_Sel[1:0] | 2’b00 | 2’b00 | 2’b10 |
| ALUCtr[2:0] | 3’b000 | 3’b001 | 3’b000 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | J | ori | sw | lw | lui | beq | jal |
| opcode | 000010 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 |
| funct | N/A | | | | | | |
| RegDst[1:0] | X | 2’b00 | 2’b00 | X | X | 2’b00 | 2’b10 |
| ALUSrc | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| MemtoReg[1:0] | X | 0 | 1 | X | X | 0 | 2’b10 |
| RegWrite | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Branch | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| J\_Sel[1:0] | 2’b01 | 2’b00 | 2’b00 | 2’b00 | 2’b00 | 2’b00 | 2’b10 |
| ExtOp[1:0] | 2’b00 | 2’b10 | 2’b00 | 2’b00 | 2’b00 | 2’b01 | 2’b00 |
| MemRead | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| PC\_Sel[1:0] | 2’b01 | 2’b00 | 2’b00 | 2’b00 | 2’b00 | 2’b01 | 2’b01 |
| ALUOp[2:0] | 3’b000 | 3’b010 | 3’b000 | 3’b000 | 3’b000 | 3’b000 | 3’b000 |

1. 冲突控制器

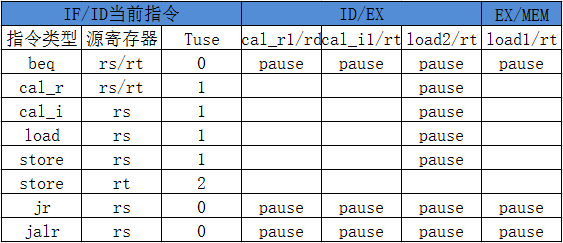
(1)模块接口

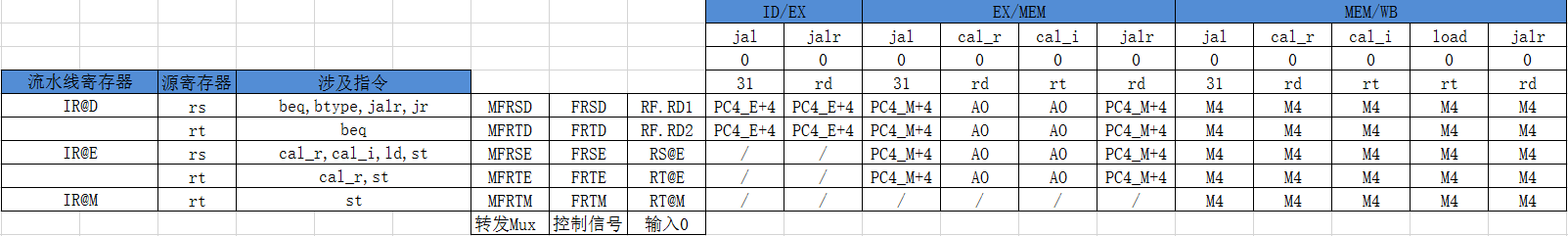
|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| Conflict\_manager.v | Conflict\_manager(IR\_D, IR\_E, IR\_M, IR\_W, FRSD, FRTD, FRSE, FRTE, FRTM, stall);  input [31:0] IR\_D;  input [31:0] IR\_E;  input [31:0] IR\_M;  input [31:0] IR\_W;  input busy;  input start;  output [1:0] FRSD;  output [1:0] FRTD;  output [1:0] FRSE;  output [1:0] FRTE;  output [1:0] FRTM;  output stall; |

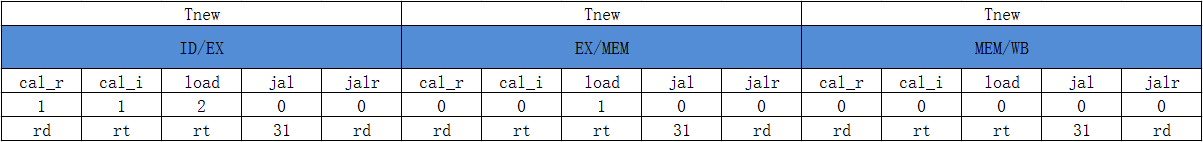
（2）功能定义

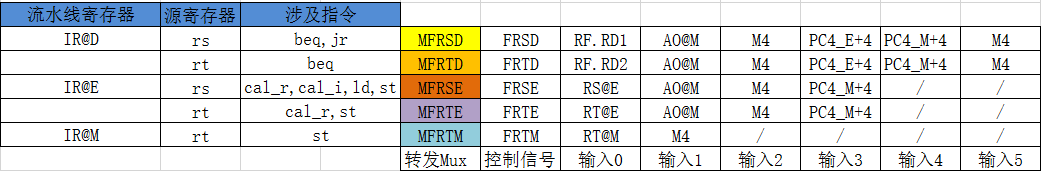
|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生暂停信号 | 暂停信号stall |
| 2 | 产生转发信号 | 5个转发信号 |

暂停机制



转发机制





1. 测试代码

lui $3,0xf0e3

ori $4,$0,12

addi $5,$4,15

nop

nop

#test sll sllv sra srav srl srlv

addi $6,$3,1

srl $6,$6,2 #R\_M\_RT

addi $8,$4,-10

srlv $7,$6,$8 #R\_M\_RS

srl $8,$7,3 #R\_M\_RT

sll $8,$7,2 #R\_W\_RS

sra $9,$8,1

ori $8,$0,1

sllv $9,$9,$8

li $9,10

srav $9,$3,$9

#test slt slti sltu sltiu

slt $10,$9,$8

sltu $11,$9,$8

slt $12,$11,$10 #R\_W\_RT

bgtz $12,label #test bgtz

slti $12,$12,3

nop

nop

label:

addi $1,$0,2

bgez $12,label2 #I\_M\_RS test bgez

addi $1,$1,1

nop

label2:

#test store and load

sw $3,4($0)

sw $4,8($0)

sw $5,12($0)

lw $6,12($0)

sltiu $7,$6,0xefe #Ld\_M\_RS

nor $8,$6,$7 #I\_M\_RT

xor $9,$8,$7 #I\_W\_RT

xori $10,$9,0xfe10

sh $10,14($0) #test sh

sb $10,9($0) #test sb

add $10,$10,$9

slti $11,$10,10

sb $11,10($0)

lh $12,10($0) #test lh

lbu $13,11($0) #test lbu

lb $13,9($0) #test lb

lh $14,6($0) #test lh

lhu $15,6($0) #test lhu

lb $16,13($0)

li $6,4

lh $17,10($6)

bgez $17,label3 #test bgez

mult $4,$5

mflo $6

addi $6,$5,1

nop

label4:

nor $6,$4,$5

nor $7,$3,$6

and $8,$6,$7

or $9,$8,$7

sub $9,$9,$8

addi $10,$9,12

addi $12,$10,12

addi $15,$0,0x3130

jal label5

addi $12,$0,12

div $12,$10

li $16,-1

bltz $16,label6

nop

label6:

sll $0,$0,0

nop

j end

nop

label5:

jalr $12,$15

nop

jr $31

lui $11,1

nop

div $15,$12

jr $12

nop

label3:

#test mult div multu divu mfhi mflo mthi mtlo

mflo $7

mult $7,$5 #R\_M\_RS

addi $6,$0,5

mflo $8

div $7,$6 #R\_W\_RT

mflo $9

divu $7,$9

mfhi $10

mflo $11

li $11,-2

li $12,-3

mult $11,$12

mflo $13

li $7,2

multu $11,$7

divu $11,$13

mfhi $15

mflo $16

add $17,$16,$15

div $17,$16

mthi $17

sub $18,$17,$17

mflo $18

mfhi $18

sll $18,$17,10

sra $19,$18,10

multu $18,$19

mfhi $21

lw $8,4($0)

lh $9,6($0)

mult $8,$9

bltz $21,label4

nop

nop

end:

addi $15,$0,1

nop

预期输出

$ 3 <= f0e30000

$ 4 <= 0000000c

$ 5 <= 0000001b

$ 6 <= f0e30001

$ 6 <= 3c38c000

$ 8 <= 00000002

$ 7 <= 0f0e3000

$ 8 <= 01e1c600

$ 8 <= 3c38c000

$ 9 <= 1e1c6000

$ 8 <= 00000001

$ 9 <= 3c38c000

$ 9 <= 0000000a

$ 9 <= fffc38c0

$10 <= 00000001

$11 <= 00000000

$12 <= 00000001

$12 <= 00000001

$ 1 <= 00000002

$ 1 <= 00000003

\*00000004 <= f0e30000

\*00000008 <= 0000000c

\*0000000c <= 0000001b

$ 6 <= 0000001b

$ 7 <= 00000001

$ 8 <= ffffffe4

$ 9 <= ffffffe5

$10 <= ffff01f5

\*0000000e <= 01f5

\*00000009 <= f5

$10 <= ffff01da

$11 <= 00000001

\*0000000a <= 01

$12 <= 00000001

$13 <= 00000000

$13 <= fffffff5

$14 <= fffff0e3

$15 <= 0000f0e3

$16 <= 00000000

$ 6 <= 00000004

$17 <= 000001f5

$ 7 <= 00000144

$ 6 <= 00000005

$ 8 <= 0000222c

$ 9 <= 00000040

$10 <= 00000004

$11 <= 00000005

$11 <= fffffffe

$12 <= fffffffd

$13 <= 00000006

$ 7 <= 00000002

$15 <= 00000002

$16 <= 2aaaaaaa

$17 <= 2aaaaaac

$18 <= 00000000

$18 <= 00000001

$18 <= 2aaaaaac

$18 <= aaaab000

$19 <= ffeaaaac

$21 <= aa9c771c

$ 8 <= f0e30000

$ 9 <= fffff0e3

$ 6 <= ffffffe0

$ 7 <= 0000001f

$ 8 <= 00000000

$ 9 <= 0000001f

$ 9 <= 0000001f

$10 <= 0000002b

$12 <= 00000037

$15 <= 00003130

$31 <= 000030fc

$12 <= 0000000c

$12 <= 00003124

$11 <= 00010000

$16 <= ffffffff

$15 <= 00000001

1. 思考题

1.为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI，LO寄存器？

我们是在模拟乘除法部件，而在硬件实现中，乘除法部件是比较复杂的，这需要一个独立的部件，同时乘除法部件中还有Hi和Lo寄存器，另外乘除法执行需要更长周期，需要设置乘除槽，这些都与ALU有很大不同，所以需要单独部件。

Hi，Lo寄存器不属于32个通用寄存器，它们存储特定的值，需要特定的指令来对其进行读写操作，为了使它们不被其他指令干扰，保持其独立性，应该设置独立的Hi，Lo寄存器。

2.参照你对延迟槽的理解，试解释“乘除槽”。

延迟槽的设计是因为当执行跳转指令时，PC为了确定下一条指令的位置，需要暂停取指令一个周期，这样会导致一个时间周期的浪费，而延迟槽的设定就是为了使这个时间周期利用起来而不被浪费掉。

乘除槽的设计也有类似效果，因为乘除运算部件执行乘除法需要5个甚至10个时间周期，而为了使执行乘除法时的时间周期不被浪费，可以设置乘除槽，使CPU能够执行除了乘除相关指令的其他指令。

3.为何上文文末提到的lb等指令使用的数据扩展模块应在 MEM/WB 之后,而不能在 DM 之后?

因为M阶段涉及到了读写寄存器，本来M级的时间就比较长了，将数据拓展模块放在W阶段有利于各阶段时间的平衡。流水线CPU的时间周期是根据各级流水线中耗时最长的那个决定的，所以不应该让M级的时间更长。

4.举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑C语言中字符串的情况）

假设内存中存储了一段英文字符串，这时需要对这个字符串中的每一个字符进行单独操作，因为一个英文字符的长度是一个字节，如果是按字节访问，则能够直接访问然后进行操作，但是如果是按字访问，那么一次取出的是一个字即4个字符，当进行操作时还要确定每个字符是什么，无疑会浪费时间，造成性能上的损耗。

5.如何概括你所设计的CPU的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

我的CPU的设计风格是规划者(planner)型。

将指令进行大致分类，如cal\_r, cal\_i, btype, j, jal, jr, jalr等类型，减少代码量；

使用`define；

命名尽量统一，如XXX\_D,XXX\_E,XXX\_M,XXX\_W等。

对各种情况进行分类，同类的写在一起，格式统一，便于观察和更改。

6. 你对流水线CPU设计风格有何见解？

我认为规划者型比较好，设计与实现分离，使思路更加清晰，错误率低；而且代码工整，各个部分比较清晰，修改时思路容易思考，但是修改工作量略大；另外规划者型可以通过脚本自动生成。