计算机组成原理P4实验报告

彭杰奇15061169

1. 数据通路设计
2. IM模块

（1）基本描述

IM是指令存储模块，由一个32bit×1024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 [6:2] Addr; // 输入的指令地址  output [31:0] Instr; // 输出的指令 |

表1 IM模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[6:2] | I | 输入的指令地址 |
| Instr[31:0] | O | 输出的指令 |

（3）功能定义

表2 IM功能定义

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

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(PCSrc,Imm32,index,Jr,Clk,Reset,Instr,PCOut);  input [1:0] PCSrc; //判断当前指令是beq、jal、jr指令中的哪一条  input [31:0] Imm\_32; // 若为beq指令，输入的是需要进行移位计算的立即数；若为jr指令，输入的是GRF[rs]中保存的值  input [25:0] index; //若为jal指令，输入需要进行处理的26位立即数  input Jr; // 当前指令是否为jr信号  input Clk; // 时钟信号  input Reset; // 复位信号，1:有效，0:无效  output [31:0] Instr; // 当前指令输出  output [31:0] PCOut; // 若为jal指令，需要保存在GRF[31]中的PC+4的值 |

表3 IFU模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PCSrc[1:0] | I | 判断当前指令是beq、jal、jr指令中的哪一条  2’b00: 若jr为1 为jr指令  若jr为0 正常  2’b01: beq指令  2’b11: jal指令 |
| Imm\_32[31:0] | I | 若为beq指令，输入的是需要进行移位计算的立即数  若为jr指令，输入的是GRF[rs]中保存的值 |
| Index[25:0] | I | 若为jal指令，输入需要进行处理的26位立即数 |
| Jr | I | 当前指令是否为jr信号 |
| Reset | I | 复位信号，1:有效，0:无效 |
| Clk | I | 时钟信号 |
| Instr[31:0] | O | 当前指令输出 |
| PCOut[31:0] | O | 若为jal指令，需要保存在GRF[31]中的PC+4的值 |

（3）功能定义

表4 IFU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 | 计算下一条指令地址 | PCSrc == 2’b00 时，  若Jr = 0，PC 🡨 PC + 4；  若Jr = 1，PC 🡨Imm\_32:  PCSrc == 2’b01 时，PC 🡨 PC + 4 + Imm\_32||0^2  PCSrc == 2’b10 时，PC 🡨 PC[31:28]||index||0^2； |

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，有RegWrite和J\_Sel两个写使能信号，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,J\_Sel,RData1,RData2);  input [4:0] rs; // rs寄存器的地址  input [4:0] rt; // rt寄存器的地址  input [4:0] wr; //要写入的寄存器的地址  input [31:0] WData; //要写入的数据  input Clk; //时钟信号  input Reset; //复位信号  input RegWrite; //一般写使能信号，1:有效，0:无效  input J\_Sel; // jal指令的写使能信号，1:有效，0:无效  output [31:0] RData1; // rs寄存器的值  output [31:0] RData2; // rt寄存器的值 |

表5 GRF模块接口

|  |  |  |
| --- | --- | --- |
| 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 |
| J\_Sel | I | jal指令的写使能信号，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

表6 GRF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  J\_Sel有效时，(GPR[31]) 🡨WData |
| 3 | 清零 | 复位信号有效时，GRF中所有寄存器都清零 |

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,Zero,ALUResult);  input [31:0] A1; //第一个运算数  input [31:0] A2; //第二个运算数  input [2:0] ALUCtr; // ALU控制信号  output Zero; //输入数据是否相等  output [31:0] ALUResult; // ALU运算结果 |

表7 ALU模块接口

|  |  |  |
| --- | --- | --- |
| 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 |
| Zero | O | 输入数据是否相等  1:相等  2:不相等 |
| ALUResult[31:0] | O | ALU运算结果 |

（3）功能定义

表8 ALU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 |
| 5 | 等于判断 | Zero 🡨 (A1-A2)==0?1:0 |

1. DM模块

（1）基本描述

DM模块为数据存储器，由一个32bit \* 32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 [6:2] Addr; //读/写DM的地址  input [31:0] Din; //要写入DM的数据  input MemWrite; //写DM的控制信号  input MemRead; //读DM的控制信号  input Clk; //时钟信号  input Reset; //复位信号  output [31:0] Dout; //从DM读出的数据 |

表9 DM模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[6:2] | 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 |
| 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

表10 DM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) |
| 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 |

1. 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,Imm32);  input [15:0] Imm\_16; //要扩展的16位立即数  input [1:0] ExtOp; //扩展方式选择信号  output [31:0] Imm\_32; //扩展后的32位立即数 |

表11 EXT模块接口

|  |  |  |
| --- | --- | --- |
| 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

表12 EXT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 |

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(opcode,funct,RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,MemRead,ExtOp,n\_PCSel,J\_Sel,Jr,ALUCtr);  input [5:0] opcode;  input [5:0] funct;  output RegDst;  output ALUSrc;  output MemtoReg;  output RegWrite;  output MemWrite;  output MemRead;  output [1:0] ExtOp;  output n\_PCSel;  output J\_Sel;  output Jr;  output [2:0] ALUCtr; // |

表13 Controller模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| opcode[5:0] | I | 指令中的opcode,即[31:26]位 |
| funct[5:0] | I | 指令中的funct,即[5:0]位 |
| RegDst | O | 寄存器写入端地址控制  1:选择rd字段  0:选择rt字段 |
| ALUSrc | O | ALU输入端A2选择  1:选择Imm\_32  0:选择RD2 |
| MemtoReg | O | 寄存器堆写入端WD选择  0:来自ALU输出  1:来自DM输出 |
| 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 |
| nPC\_Sel | O | 判断是否为beq指令 |
| J\_Sel | O | 判断是否为jal指令 |
| Jr | O | 判断是否为jr指令 |
| ALUCtr[2:0] | O | ALU控制信号 |

（3）功能定义

表14 Controller功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | addu指令 | 当前指令为addu时，RegDst、RegWrite信号为1，其他全为0 |
| 2 | subu指令 | 当前指令为subu时，RegDst、RegWrite、ALUCtr[1]信号为1，其他全为0 |
| 3 | ori指令 | 当前指令为ori时，ALUSrc、RegWrite、ALUCtr[0]、ALUCtr[1]、ExtOp[1]信号为1，其他全为0 |
| 4 | lw指令 | 当前指令为lw时，ALUSrc、MemtoReg、RegWrite、MemRead信号为1，其他全为0 |
| 5 | sw指令 | 当前指令为sw时，ALUSrc、MemWrite信号为1，其他全为0 |
| 6 | beq指令 | 当前指令为beq时，nPC\_Sel、ALUCtr[1]信号为1，其他全为0 |
| 7 | lui指令 | 当前指令为lui时，ALUSrc、RegWrite、ExtOp[0]信号为1，其他全为0 |
| 8 | jal指令 | 当前指令为jal时，J\_Sel信号为1，其他为0 |
| 9 | jr指令 | 当前指令为jr时，单独调整Jr信号为1，RegDst、RegWrite信号为1，其他全为0 |

1. Controller真值表

表15 Controller真值表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | Subu | addu | Jr | ori | lw | sw | beq | lui | jal |
| opcode | 000000 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 |
| funct | 100011 | 100001 | 001000 | N/A | | | | | |
| RegDst | 1 | 1 | 1 | 0 | 0 | X | X | 0 | X |
| ALUSrc | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | X |
| MemtoReg | 0 | 0 | 0 | 0 | 1 | X | X | 0 | X |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| nPC\_Sel | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| J\_Sel | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Jr | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| ExtOp[1] | X | X | X | 1 | 0 | 0 | 0 | 0 | X |
| ExtOp[0] | X | X | X | 0 | 0 | 0 | 0 | 1 | X |
| MemRead | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ALUOp[1] | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| ALUOp[0] | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

1. ALU控制器设计

ALU控制器是控制器的一部分，它利用funct[5:0]的值和过渡信号ALUOp[1:0]得到控制ALU运算的ALU控制信号ALUCtr[1:0]

表16 ALU控制器真值表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ALUOp | Funct | ALUCtr | 运算 | 描述 |
| 00 | X | 000 | 加法 | 针对lw、sw、lui、jal指令 |
| 01 | X | 001 | 减法 | 针对beq指令 |
| 11 | X | 011 | 按位或 | 针对ori指令 |
| 10 | 100001(addu) | 000 | 加法 | 针对addu指令 |
| 10 | 100011(subu) | 001 | 减法 | 针对subu指令 |
| 10 | 001000(jr) | 000 | 加法 | 针对jr指令 |

1. 测试程序

lui $6, 1

addu $8, $6, $0

addu $8, $6, $8

subu $9, $8, $6

jal label

beq $6, $8, label2

ori $6, $0, 8

label:

subu $8, $8, $6

ori $1, $0, 4

sw $8, 0($1)

sw $6, 4($1)

jr $31

label2:

ori $9, $8, 1

lw $6, 0($1)

lw $8, 4($1)

subu $7, $8, $9

addu $5, $6, $9

lui $5,10

预期结果

$ 6 <= 00010000

$ 8 <= 00010000

$ 8 <= 00020000

$ 9 <= 00010000

$31 <= 00003014

$ 8 <= 00010000

$ 1 <= 00000004

\*00000004 <= 00010000

\*00000008 <= 00010000

$ 9 <= 00010001

$ 6 <= 00010000

$ 8 <= 00010000

$ 7 <= ffffffff

$ 5 <= 00020001

$ 5 <= 000a0000

1. 思考题

1.根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

因为我们输入DM中的是32位信号，而地址信号addr需要10位，地址又是4的倍数，所以最后两位是0，而DM是每次移动一个子单元，所以我们不截取最后两位，而截取[11:2]。addr信号来自ALU运算结果。

2.在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

PC(IFU)、GRF、DM

因为Reset为高电平的时候，电路需要复位初始化，所以PC要复位为初始地址，重取第一条指令，电路初始化的时候，电路中的寄存器和存储器中存储的信息也要清零。

3.列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

1.利用if-else或case

2.利用assign语句

3.利用宏定义

|  |  |
| --- | --- |
| 编码方式 | 代码举例 |
| case(if-else) | always@(opcode) begin  case(opcode)  6’b000000: ALUOp = 2’b10;  ……;  6’b001101: ALUOp = 2’b11;  ……;  6’b100011: ALUOp = 2’b00;  ……;  6’b101011: ALUOp = 2’b00;  ……;  6’b000100: ALUOp = 2’b01;  ……;  6’b001111: ALUOp = 2’b00;  ……;  6’b000011: ALUOp = 2’b01;  ……;  endcase  end |
| assign | assign ALUOp = (opcode==6’b000000)?2’b10:  (opcode==6’b001101) ?2’b11:  (opcode==6’b100011) ?2’b00:  (opcode==6’b101011) ?2’b00:  (opcode==6’b000100) ?2’b01:  (opcode==6’b001111) ?2’b00:  (opcode==6’b000011) ?2’b01:  2’b00 |
| 宏定义 | `define ALUOPTMP (opcode==6’b000000)?2’b10:……2’b00  assign ALUOp = `ALUOPTMP; |

4.根据你所列举的编码方式，说明他们的优缺点。

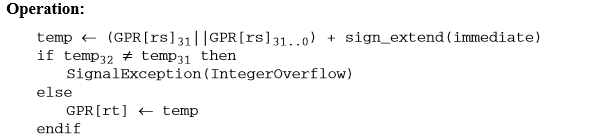
if-else或case的编码方式便于查看同一条指令的不同信号，但是对于同一个信号是由哪些指令怎么控制查看不直观

assign的编码方式便于查看同一个信号由不同指令怎么控制，但是对于同一条指令对不同信号的控制查看不方便

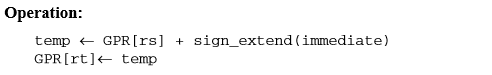
宏定义用宏名代替部分代码，可以提高程序的可移植性和可读性，但是宏定义用宏名代替代码时只做简单的置换，不做语法检查，预处理时照样代入而不管含义是否正确，只有在编译已被展开宏展开的源程序时才会报错。

5. C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分

addi:



addiu:



在operation部分中，addi和add是会计算是否溢出的，若temp32和temp31不相等的话会检测到溢出，那么会返回溢出错误， 而如果忽略溢出的话，那么if块中的就不会执行，temp会直接输入GPR[rd]中，而addu和addiu本来就不用检测溢出，temp直接输入GOR[rd]，所以这样addi和addiu，add和addu就没有区别了。

6.根据自己的设计说明单周期处理器的优缺点。

优点：

1.构造相对简单；

2.因为是单周期，一个周期内执行一条指令，不会发生数据冲突；

缺点：

1.单周期处理器需要足够长的周期来完成最慢的指令，及时大部分指令的速度都非常快；

2.需要3个加法器，而加法器是相对占用芯片面积的电路，尤其是速度比较快时；

3.采用独立的指令存储器和数据存储器，这在实际系统中不现实；

7.简要说明jal、jr和堆栈的关系。

MIPS使用jal指令调用一个函数，使用jr指令从函数返回。在调用函数时，调用函数会将返回地址PC+4储存在$ra寄存器中，与此同时使用jal指令跳转到被调动函数入口，被调用函数会创建栈空间来存储一个或多个寄存器的值，然后将寄存器的值存储在栈中，使用寄存器执行函数，再从栈中恢复寄存器的原始值，回收栈空间，在函数返回时，需要将保存数据出栈，然后执行jr $ra 来立即返回到jal后面的指令。