计算机组成原理 P4 实验报告

彭杰奇 15061169

一、 数据通路设计

1. 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]

2. 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；

3. 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

4. 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

5. 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

6. 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

二、 控制器设计

1. 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

2. 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

3. 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

三、 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

四、 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:

Operation:

```
temp ← (GPR[rs]31||GPR[rs]31..0) + sign_extend(immediate)
if temp32 ≠ temp31 then
   SignalException(IntegerOverflow)
else
   GPR[rt] ← temp
endif
```

addiu:

Operation:

```
temp ← GPR[rs] + sign_extend(immediate)
GPR[rt]← temp
```

在 operation 部分中，addi 和 add 是会计算是否溢出的，若 $temp_{32}$ 和 $temp_{31}$ 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。