计算机组成原理 P6 实验报告

彭杰奇 15061169

一、 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

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(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

2. 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

三、 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

IF/ID当前指令			ID/EX			EX/MEM
指令类型	源寄存器	Tuse	cal_r1/rd	cal_i1/rt	load2/rt	load1/rt
beq	rs/rt	0	pause	pause	pause	pause
cal_r	rs/rt	1			pause	
cal_i	rs	1			pause	
load	rs	1			pause	
store	rs	1			pause	
store	rt	2				
jr	rs	0	pause	pause	pause	pause
jalr	rs	0	pause	pause	pause	pause

转发机制

						ID/EX		EX/MEM				MEM/WB				
						jal	jalr	jal	cal_r	cal_i	jalr	jal	cal_r	cal_i	load	jalr
						0	0	0	0	0	0	0	0	0	0	0
流水线寄存器	源寄存器	涉及指令				31	rd	31	rd	rt	rd	31	rd	rt	rt	rd
IR@D	rs	beq, btype, jalr, jr	MFRSD	FRSD	RF. RD1	PC4_E+4	PC4_E+4	PC4_M+4	AO	AO	PC4_M+4	M4	M4	M4	M4	M4
	rt	beq	MFRTD	FRTD	RF. RD2	PC4_E+4	PC4_E+4	PC4_M+4	AO	AO	PC4_M+4	M4	M4	M4	M4	M4
IR@E	rs	cal_r, cal_i, ld, st	MFRSE	FRSE	RS@E	/	/	PC4_M+4	AO	AO	PC4_M+4	M4	M4	M4	M4	M4
	rt	cal_r, st	MFRTE	FRTE	RT@E	/	/	PC4_M+4	AO	AO	PC4_M+4	M4	M4	M4	M4	M4
IR@M	rt	st	MFRTM	FRTM	RT@M	/	/	/	/	/	/	M4	M4	M4	M4	M4
			转发Mux	控制信号	输入0											

Tnew					Tnew					Tnew				
ID/EX					EX/MEM					MEM/WB				
cal_r	cal_i	load	jal	jalr	cal_r	cal_i	load	jal	jalr	cal_r	cal_i	load	jal	jalr
1	1	2	0	0	0	0	1	0	0	0	0	0	0	0
rd	rt	rt	31	rd	rd	rt	rt	31	rd	rd	rt	rt	31	rd

流水线寄存器	源寄存器	涉及指令								
IR@D	rs	beq, jr	MFRSD	FRSD	RF. RD1	AO@M	M4	PC4_E+4	PC4_M+4	M4
	rt	beq	MFRTD	FRTD	RF. RD2	AO@M	M4	PC4_E+4	PC4_M+4	M4
IR@E	rs	cal_r, cal_i, ld, st	MFRSE	FRSE	RS@E	AO@M	M4	PC4_M+4	/	/
	rt	cal_r, st	MFRTE	FRTE	RT@E	AO@M	M4	PC4_M+4	/	/
IR@M	rt	st	MFRTM	FRTM	RT@M	M4	/	/	/	/
			转发Mux	控制信号	输入0	输入1	输入2	输入3	输入4	输入5

四、 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

五、 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。