Project6 Verilog 完成流水线处理器开发（Plus）

一、顶层设计

1、顶层视图：

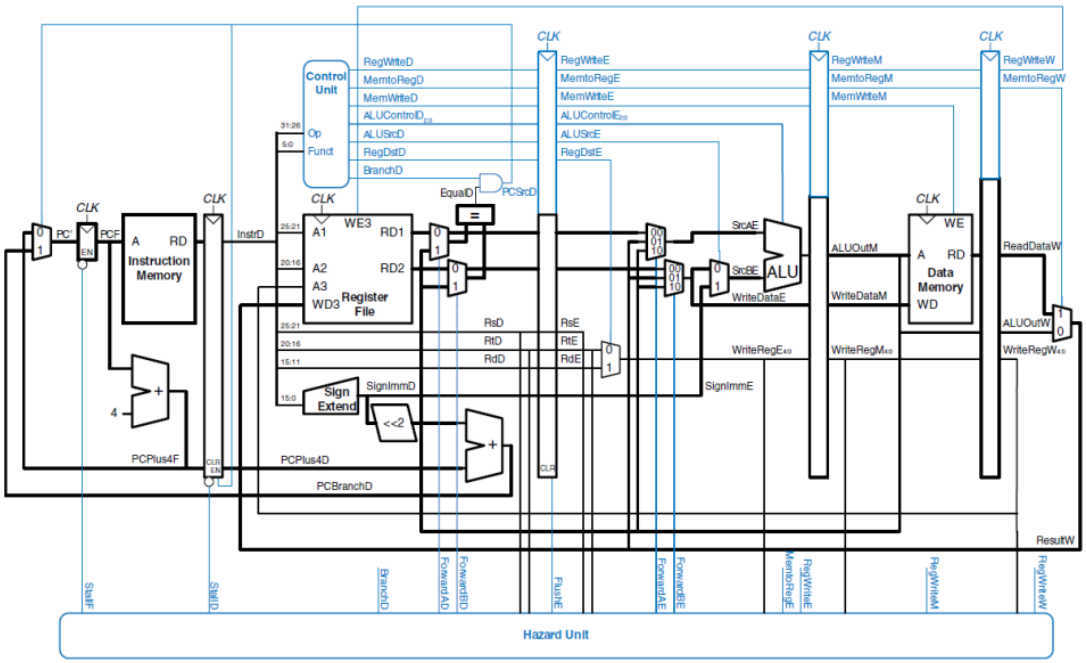

Figure 7.58 Pipelined processor with full hazard handling

2、支持指令集：

MIPS-C3={LB、LBU、LH、LHU、LW、SB、SH、SW、
ADD、ADDU、SUB、 SUBU、 MULT、 MULTU、 DIV、DIVU、
SLL、 SRL、 SRA、 SLLV、SRLV、SRAV、
AND、OR、XOR、NOR、
ADDI、ADDIU、ANDI、ORI、XORI、LUI、
SLT、SLTI、SLTIU、SLTU、
BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、
J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}

3、顶层文件接口定义：

文件	模块接口定义

mips.v	module mips(clk,reset);
	input clk; //clock
	input reset; //reset

二、数据通路设计

1、数据通路架构：

	部件	描述	输入	输入来源				MUX	MUX控制信号	lw	sw	addu	subu	ori	lui	beq	j	jal	jr
F级功能部件	PC	程序计数器																	
	ADD4	完成PC+4		PC						PC	PC	PC	PC	PC	PC	PC	PC	PC	PC
	IM	指令存储器		PC						PC	PC	PC	PC	PC	PC	PC	PC	PC	PC
D级更新PC	PC			ADD4						ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	
D级流水线寄存器	IR@D	传递指令		IM						IM	IM	IM	IM	IM	IM	IM	IM	IM	IM
	PC4@D	下一条指令地址		ADD4												ADD4	ADD4	ADD4	
D级功能部件	RF	寄存器堆	A1	IR@D[rs]						IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]			IR@D[rs]
			A2	IR@D[rt]							IR@D[rt]	IR@D[rt]	IR@D[rt]			IR@D[rt]			
	EXT	立即数扩展		IR@D[i16]						IR@D[i16]	IR@D[i16]			IR@D[i16]	IR@D[i16]				
	CMP	比较2个数	D1	RF.RD1												RF.RD1			
			D2	RF.RD2												RF.RD2			
	NPC	计算下条地址	PC4	PC4@D												PC4@D	PC4@D	PC4@D	
			I26	IR@D[i16]	IR@D[i26]											IR@D[i16]	IR@D[i26]	IR@D[i26]	
E级更新PC	PC			NPC	RF.RD1			MUX_PC	Pcsel							NPC	NPC	NPC	RF.RD1
E级流水线寄存器	IR@E	传递指令		IR@D						IR@D	IR@D	IR@D	IR@D	IR@D	IR@D			IR@D	
	PC4@E	下一条指令地址		PC4@D														PC4@D	
	RS@E	RF的RS值		RF.RD1						RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1				
	RT@E	RF的RT值		RF.RD2							RF.RD2	RF.RD2	RF.RD2						
	EXT@E	扩展后的立即数		EXT						EXT	EXT			EXT	EXT				
E级功能部件	ALU	算数逻辑运算	A	RS@E						RS@E	RS@E	RS@E	RS@E	RS@E	RS@E				
			B	RT@E	EXT@E			MUX_ALUb	ALU_bsel	EXT@E	EXT@E	RT@E	RT@E	EXT@E	EXT@E				
M级流水线寄存器	IR@M	传递指令		IR@E						IR@E	IR@E	IR@E	IR@E	IR@E	IR@E			IR@E	
	PC4@M	下一条指令地址		PC4@E														PC4@E	
	AO@M	ALU读出结果		ALU						ALU	ALU	ALU	ALU	ALU	ALU				
	RT@M	RF的RT值		RT@E							RT@E								
M级功能部件	DM	数据存储器	A	AO@M						AO@M	AO@M								
			WD	RT@M							RT@M								
W级流水线寄存器	IR@W	传递指令		IR@M						IR@M		IR@M	IR@M	IR@M	IR@M			IR@M	
	PC4@W	下一条指令地址		PC4@M														PC4@M	
	AO@W	ALU读出结果		AO@M								AO@M	AO@M	AO@M	AO@M				
	DR@W	DM读出结果		DM						DM									
W级功能部件	RF	寄存器堆	A3	IR@W[rd]	IR@W[rt]	0x1F		MUX_GPRwa	RF_WAsel	IR@W[rt]		IR@W[rd]	IR@W[rd]	IR@W[rt]	IR@W[rt]			0x1F	
			WD	AO@W	DR@W	EXT	PC4@W	MUX_GPRwd	RF_WDsel	DR@W		AO@W	AO@W	AO@W	AO@W			PC4@W	

		描述	lw	sw	addu	subu	ori	lui	beq	j	jal	jr	MUX	0	1	2	MUX控制信号
PC		程序计数器	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	NPC	NPC	RF.RD1	MPC	ADD4	NPC	RF.RD1	PCsel
IM		指令存储器	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC		PC			
ADD4		完成PC+4	PC	PC	PC	PC	PC	PC	PC	PC	PC			PC			
D级	IR	传递指令	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM		IM			
	NPC	下一条指令地址	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4			ADD4			
RF	A1	第1个源寄存器编号	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D			IR[rs]@D		IR[rs]@D			
	A2	第2个源寄存器编号		IR[rt]@D	IR@D[rt]	IR@D[rt]			IR[rt]@D					IR[rt]@D			
EXT		立即数扩展	IR[i16]@D	IR[i16]@D			IR[i16]@D	IR[i16]@D						IR[i16]@D			
NPC	PC4	计算下条地址							PC4@D	PC4@D	PC4@D			PC4@D			
	I26								IR[i16]@D	IR[i26]@D	IR[i26]@D			IR[i26]@D			
CMP	D1	比较2个数							RF.RD1					RF.RD1			
	D2								RF.RD2					RF.RD2			
E级	V1	RF的第1个寄存器输出值	RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1						RF.RD1			
	V2	RF的第2个寄存器输出值		RF.RD2	RF.RD2	RF.RD2								RF.RD2			
	A1	第1个源寄存器编号	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D						IR[rs]@D			
	A2	第2个源寄存器编号		IR[rt]@D	IR[rt]@D	IR[rt]@D								IR[rt]@D			
	A3	目的寄存器编号	IR[rt]@D		IR[rd]@D	IR[rd]@D	IR[rt]@D	IR[rt]@D			0x1F		MA3E	IR[rt]@D	IR[rd]@D	0x1F	RFA3sel
	E32	EXT的32位扩展结果	EXT	EXT			EXT	EXT						EXT			
	PC4	下一条指令地址									PC4@D			PC4@D			
ALU	A	算数逻辑运算	V1@E	V1@E	V1@E	V1@E	V1@E	V1@E						V1@E			
	B		E32@E	E32@E	V2@E	V2@E	E32@E	E32@E					MALUB	V2@E	E32@E		ALUBsel
M级	V2	RF的第2个寄存器输出值		V2@E										V2@E			
	A2	第2个源寄存器编号		A2@E										A2@E			
	AO	ALU读出结果	ALU	ALU	ALU	ALU	ALU	ALU						ALU			
	A3		A3@E		A3@E	A3@E	A3@E	A3@E			A3@E			A3@E			
	PC4	下一条指令地址									PC4@E			PC4@E			
DM	A	写入地址	AO@M	AO@M	AO@M	AO@M	AO@M	AO@M						AO@M			
	WD	写入值		V2@M										V2@M			
W级	A3	传递指令	A3@M		A3@M	A3@M	A3@M	A3@M			A3@M			A3@M			
	PC4	下一条指令地址									PC4@M			PC4@M			
	AO	ALU读出结果	AO@M		AO@M	AO@M	AO@M	AO@M						AO@M			
	DR	DM输出值	DM											DM			
RF	A3	写入寄存器编号	A3@W		A3@W	A3@W	A3@W	A3@W			A3@W			A3@W			
	WD	写入值	DR@W		AO@W	AO@W	AO@W	AO@M			PC4@W		MRFWD	AO@W	DR@W	PC4@W	RFWDsel

		lw	sw	addu	subu	ori	lui	beq	j	jal	jr	MUX	0	1	2
PC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	NPC	NPC	RF.RD1	MPC	ADD4	NPC	MF_PC
CMP	D1							RF.RD1				MF_CMP_D1			
	D2							RF.RD2				MF_CMP_D2			
E级	A3	IR[rt]@D		IR[rd]@D	IR[rd]@D	IR[rt]@D	IR[rt]@D			0x1F		MA3E	IR[rt]@D	IR[rd]@D	0x1F
ALU	B	E32@E	E32@E	V2@E	V2@E	E32@E	E32@E					MALUB	MF_ALU_B	E32@E	
M级	V2		V2@E									MF_M_V2			
DM	WD		V2@M									MF_DM_WD			
RF	WD	DR@W		AO@W	AO@W	AO@W	AO@M			PC4@W		MRFWD	AO@W	DR@W	PC4@W

2、PC

（1）基本描述：

PC 用寄存器实现，具有复位功能。

起始地址：0x0000_3000。

（2）文件接口定义：

文件	模块接口定义
PC.v	module PC(Clock,Reset,Hold,NextPC,PC); input Clock, //clock input Reset, //reset input Hold, //stay the same input [31:0] NextPC, //next PC input output reg[31:0] PC //new PC

（3）功能定义：

序号	功能名称	功能描述
1	复位	当Reset有效且时钟上升沿来临时,PC被设置为0x0000_3000
2	更新PC的值	在时钟上升沿来临时，将NextPC写入到PC中
3	保持PC的值	当Reset无效、Hold有效且时钟上升沿来临时，PC保持不变

3、IM

（1）基本描述：

IM 容量为 4KB（32bit×1024 字）。

生成相应数量的 32 位寄存器的阵列。

采用$readmemh 指令将指令读入到 IM 中去。

（2）文件接口定义：

文件	模块接口定义
IM.v	module IM(Address,Instr); input [9:0] Address, //Instruction address output [31:0] Instr //Instruction

（3）功能定义：

序号	功能名称	功能描述
1	取指令	根据PC从IM中取出指令

4、GRF

（1）基本描述：

用具有写使能的寄存器实现，寄存器总数为 32 。

0 号寄存器的值始终保持为 0。其他寄存器初始值均为 0。

（2）文件接口定义：

文件	模块接口定义
IM.v	module GRF(Clock,Reset, RegWrite, ReadReg1, ReadReg2, WriteReg, WriteData, WPC, ReadData1, ReadData2); input Clock, //clock input Reset, //reset input RegWrite, //write enable input [4:0] ReadReg1, //read reg1 input [4:0] ReadReg2, //read reg2 input [4:0] WriteReg, //write reg input [31:0] WriteData , //write data input [31:0] WPC, // PC output [31:0] ReadData1, //read data1 output [31:0] ReadData2 //read data2

（3）功能定义：

序号	功能名称	功能描述
1	复位	Reset 信号有效时，所有寄存器存储的数值清零
2	读数据	读出 ReadReg1， ReadReg2 地址对应寄存器中所存储的数据到 ReadData1， ReadData2
3	写数据	当 RegWrite 有效且时钟上升沿来临时，将 WriteData 写入 WriteReg 所对应的寄存器中

5、EXT

（1）基本描述：

扩展

（2）文件接口定义：

文件	模块接口定义
EXT.v	module EXT(Imm_16,ExtOp,Imm_32); input [15:0] Imm_16, //input 16 bit Immediate input [1:0] ExtOp, //control the extension

	output reg [31:0] Imm_32 //output 32 bit Immediate

（3）功能定义：

序号	功能名称	功能描述
1	符号扩展	将 16 位输入数据符号扩展为 32 位
2	零扩展	将 16 位输入数据符号置为低 16 位，高 16 位置 0
3	加载至高位	将 16 位输入数据符号置为高 16 位，低 16 位置 0
4	符号扩展之后，左移两位	将 16 位输入数据符号扩展为 32 位之后，左移两位

6、NPC

（1）基本描述：

计算下一个 PC 的值

（2）文件接口定义：

文件	模块接口定义
NPC.v	module NPC(Instr,pc,rs,Zero,nPCOp,npc,pc_4); input [25:0] Instr, // Instruction input [31:0] pc, // PC input [31:0] rs, // $rs input Zero, // ALU Zero input [2:0] nPCOp, // control the operation output [31:0] npc, // next PC output [31:0] pc_4 // PC + 4

（3）功能定义：

序号	功能名称	功能描述
1	计算下一 PC	计算下一 PC 的值

7、ALU

（1）基本描述：

提供 32 位加、减、或运算等功能。

可以不支持溢出（不检测溢出）。

（2）文件接口定义：

文件	模块接口定义
ALU.v	module ALU(A,B,ALUOp,Zero.Result);

	input [31:0] A, // input data A input [31:0] B, // input data B input [4:0] ALUOp, // control the ALU output Zero, // Result == 0 output [31:0] Result // result

8、DM

（1）基本描述：

DM 容量为 16KB（32bit×4096 字）。

起始地址：0x00000000

（2）文件接口定义：

文件	模块接口定义
DM.v	module DM(Clock,Reset, Memwrite, Address, WriteData ,pc,addr, ReadData); input Clock, //clock input Reset, //reset input Memwrite, //memory write enable input [9:0] Address, // address input [31:0] WriteData, //write data input [31:0] pc, // PC input [31:0] addr, // 32-bit addr output[31:0] ReadData //read data

（3）功能定义：

序号	功能名称	功能描述
1	复位	Reset 信号有效时，所有储存器存储的数值清零
2	读数据	读出地址 Address 中所存储的数据到 ReadData
3	写数据	当 Memwrite 有效且时钟上升沿来临时，将 WriteData 写入地址 Address

三、控制器设计

1、基本描述：

控制单元基于指令的 opcode 字段（$Instr_{31:26}$）和 funct 字段（$Instr_{5:0}$）计算控制信号。采用分布式译码，可以只把指令在流水级之间传递，然后在不同流水级分别实例化一个同样的控制模块进行译码即可。

2、模块定义：

信号名	方向	描述
Op [5:0]	I	用于识别指令的功能
Func [5:0]	I	用于辅助 op 来识别指令
RegDst [1:0]	O	控制写入端地址选择 00：寄存器堆写入端地址选择 Rt 字段 01：寄存器堆写入端地址选择 Rd 字段 10：寄存器堆写入端地址选择 31 号寄存器
RegWrite	O	GRF 的写使能信号 0：无效 1：把数据写入寄存器堆中对应寄存器
ALUSrc	O	控制 ALU 的操作 0：ALU 输入端 B 选择寄存器堆输出 R[rt] 1：ALU 输入端 B 选择 extend(immediate)
MemWrite	O	DM 的写使能信号 0：无效 1：数据存储器 DM 写数据（输入）
MemtoReg [1:0]	O	控制数据 lh 从 ALU 读出还是从 DM 读出 00：寄存器堆写入端数据来自 ALU 输出 01：寄存器堆写入端数据来自 DM 输出 10：寄存器堆写入端数据来自 PC+4
ExtOp [1:0]	O	EXT 功能的选择信号 00：符号扩展 01：零扩展 10：加载至高位 11：符号扩展之后，左移两位
NPCOp [3:0]	O	下一 PC 的选择信号 0000：PC = PC+4 0001：执行 beq 分支指令 0010：执行 bgez 分支指令 0011：执行 bgtz 分支指令 0100：执行 blez 分支指令 0101：执行 bltz 分支指令 0110：执行 bne 分支指令 0111：执行 j/jal 跳转指令 1000：执行 jalr/jr 分支指令
ALUOp [4:0]	O	ALU 功能的选择信号 00000：加法运算

		00001：与运算 00010：或非运算 00011：或运算 00100：减法运算 00101：异或运算 00110：符号乘 00111：无符号乘 01000：符号除 01001：无符号除 01010：逻辑左移 01011：逻辑可变左移 01100：小于置 1（有符号） 01101：小于置 1（无符号） 01110：算数右移 01111：算数可变右移 10000：逻辑右移 10001：逻辑可变右移

3、文件接口定义：

文件	模块接口定义
ctrl.v	module ctrl(Op,Func,RegDst,RegWrite,ALUSrc,MemWrite,MemReg, ExtOp,ALUOp,NPCOp); input [5:0] Op, input [5:0] Func, output [1:0] RegDst, output RegWrite, output ALUSrc, output MemWrite, output [1:0]MemtoReg, output [1:0] ExtOp, output [3:0] ALUOp, output [2:0] NPCOp

4 支持指令集：

类型	指令	序号	RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUOp	MDUOp	NPCOp
R	add	1	01	1	0	0	00	xx	0000	xxxx	0000
	addu	4	01	1	0	0	00	xx	0000	xxxx	0000
	and	5	01	1	0	0	00	xx	0001	xxxx	0000
	nor	35	01	1	0	0	00	xx	0010	xxxx	0000
	or	36	01	1	0	0	00	xx	0011	xxxx	0000
	sub	50	01	1	0	0	00	xx	0100	xxxx	0000
	subu	51	01	1	0	0	00	xx	0100	xxxx	0000
	xor	54	01	1	0	0	00	xx	0101	xxxx	0000
	sll	40	01	1	0	0	00	xx	0110	xxxx	0000
	sllv	41	01	1	0	0	00	xx	0111	xxxx	0000
	slt	42	01	1	0	0	00	xx	1000	xxxx	0000
	sltu	45	01	1	0	0	00	xx	1001	xxxx	0000
	sra	46	01	1	0	0	00	xx	1010	xxxx	0000
	srav	47	01	1	0	0	00	xx	1011	xxxx	0000
	srl	48	01	1	0	0	00	xx	1100	xxxx	0000
	srlv	49	01	1	0	0	00	xx	1101	xxxx	0000
I	addi	2	00	1	1	0	00	00	0000	xxxx	0000
	addiu	3	00	1	1	0	00	00	0000	xxxx	0000
	andi	6	00	1	1	0	00	01	0001	xxxx	0000
	ori	37	00	1	1	0	00	01	0011	xxxx	0000
	xori	55	00	1	1	0	00	01	0101	xxxx	0000
	slti	43	00	1	1	0	00	00	1000	xxxx	0000
	sltiu	44	00	1	1	0	00	00	1001	xxxx	0000
	lui	25	00	1	1	0	00	10	0000	xxxx	0000
B	beq	7	xx	0	x	0	xx	xx	xxxx	xxxx	0001
	bgez	8	xx	0	x	0	xx	xx	xxxx	xxxx	0010
	bgtz	9	xx	0	x	0	xx	xx	xxxx	xxxx	0011
	blez	10	xx	0	x	0	xx	xx	xxxx	xxxx	0100
	bltz	11	xx	0	x	0	xx	xx	xxxx	xxxx	0101
	bne	12	xx	0	x	0	xx	xx	xxxx	xxxx	0110
M-D	div	14	xx	0	x	0	xx	xx	xxxx	000	0000
	divu	15	xx	0	x	0	xx	xx	xxxx	001	0000
	mult	33	xx	0	x	0	xx	xx	xxxx	010	0000
	multu	34	xx	0	x	0	xx	xx	xxxx	011	0000
	mfhi	28	01	1	x	0	00	xx	xxxx	xxx	0000
	mflo	29	01	1	x	0	00	xx	xxxx	xxx	0000
	mthi	31	xx	0	x	0	xx	xx	xxxx	100	0000
	mtlo	32	xx	0	x	0	xx	xx	xxxx	101	0000

J	j	17	xx	0	x	0	xx	xx	xxxx	xxxx	0111
	jal	18	10	1	x	0	10	xx	xxxx	xxxx	0111
	jalr	19	01	1	x	0	10	xx	xxxx	xxxx	1000
	jr	20	xx	0	x	0	xx	xx	xxxx	xxxx	1000
Load	lb	21	00	1	1	0	01	00	0000	xxxx	0000
	lbu	22	00	1	1	0	01	00	0000	xxxx	0000
	lh	23	00	1	1	0	01	00	0000	xxxx	0000
	lhu	24	00	1	1	0	01	00	0000	xxxx	0000
	lw	26	00	1	1	0	01	00	0000	xxxx	0000
Store	sb	38	xx	0	1	1	xx	00	0000	xxxx	0000
	sh	39	xx	0	1	1	xx	00	0000	xxxx	0000
	sw	52	xx	0	1	1	xx	00	0000	xxxx	0000
	nop		00	0	0	0	00	00	0000	xxxx	0000

四、数据冒险

Tuse：指令进入 D 级后，其后的某个功能部件再经过多少 cycle 就必须要使用寄存器的值

Tnew：位于 E 级及其后各级的指令，再经过多少周期能够产生要写入寄存器的结果

暂停：Tnew＞Tuse

转发：Tnew=0 && 指令不在 W 级 ||

	Tuse	
	rs	rt
addu	1	1
subu	1	1
ori	1	
lui		
lw	1	
sw	1	2
beq	0	0
j		
jr	0	
jal		
	{0,1}	{0,1,2}

指令	功能部件	Tuse		
		E	M	W
addu	ALU	1	0	0
subu	ALU	1	0	0
ori	ALU	1	0	0
lui	ALU	1	0	0
lw	DM	2	1	0
sw				
beq				
j				
jr				
jal	PC	0	0	0

Tnew≤Tuse

rs策略矩阵

Tnew \ Tuse	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

rt策略矩阵

Tnew \ Tuse	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

转发表格

流水级	源寄存器	指令	转发MUX	控制信号		E级	M级			W级			
						jal 0/31	cal_r 0/rd	cal_i 0/rt	jal 0/31	cal_r 0/rd	cal_i 0/rt	load 0/rt	jal 0/31
D	rs	beq,jr	MFRSD	RSDsel	RF.RD1	PC_E+8	AO	AO	PC_M+8	WD	WD	WD	WD
	rt	beq	MFRTD	RTDsel	RF.RD2	PC_E+8	AO	AO	PC_M+8	WD	WD	WD	WD
E	rs	cal_r,cal_i, load,store	MFRSE	RSEsel	RS@E		AO	AO	PC_M+8	WD	WD	WD	WD
	rt	cal_r,store	MFRTE	RTEsel	RT@E		AO	AO	PC_M+8	WD	WD	WD	WD
M	rt	strore	MFRTM	RTMsel	RT@M					WD	WD	WD	WD

暂停表格

D级指令			E级(Tnew)			M级(Tnew)			W级(Tnew)		
指令	源寄存器	Tuse	cal_r 1/rd	cal_i 1/rt	load 2/rt	cal_r 0/rd	cal_i 0/rt	load 1/rt	cal_r 0/rd	cal_i 0/rt	load 0/rt
beq	rs/rt	0	S	S	S			S			
cal_r	rs/rt	1			S						
cal_i	rs	1			S						
load	rs	1			S						
strore	rs	1			S						
	rt	2			S						
jr	rs	0	S	S	S			S			

五、测试

```
    lw $1, 1234($0)
    li $2, 87654321
    addu $2, $1, $2
    li $3, 23435123
    li $4, -14353215
    multu $2, $3
    li $22, -23345454
    addu $24, $22, $2
    nop
    nop
    sllv $24, $22, $0
    beq $24, $22, skip0
    nop
    addiu $2, $2, 43222
skip0:  sw $2, 0($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $25, 1594176031
    addu $10, $25, $2
    nop
    addu $10, $25, $0
    nop
    bne $10, $25, skip1
    nop
    addiu $2, $2, 31929
skip1:  sw $2, 4($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $25, -2119359507
    addu $11, $25, $2
    xor $11, $25, $0
    nop
    nop
    beq $11, $25, skip2
    nop
    addiu $2, $2, 3010
skip2:  sw $2, 8($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $19, 1297194016
    addu $14, $19, $2
    nop
    nop
    xor $14, $19, $0
    beq $14, $19, skip3
    nop
    addiu $2, $2, 29169
skip3:  sw $2, 12($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $23, 1553809414
    addu $24, $23, $0
    nop
    or $24, $23, $2
    nop
    beq $24, $23, skip4
    nop
    addiu $2, $2, 6765
skip4:  sw $2, 16($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $11, 1462873855
    addu $25, $11, $2
    subu $25, $11, $0
    nop
    nop
    beq $25, $11, skip5
    nop
    addiu $2, $2, 24212
skip5:  sw $2, 20($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $21, -1935796249
    addu $29, $21, $0
    nop
    nop
```

```
        or $29, $21, $2
        beq $29, $21, skip6
        nop
        addiu $2, $2, 20534
skip6:  sw $2, 24($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $8, 600920159
        addu $10, $8, $0
        nop
        sllv $10, $8, $2
        nop
        beq $10, $8, skip7
        nop
        addiu $2, $2, 18367
skip7:  sw $2, 28($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $10, 245016401
        addu $16, $10, $0
        or $16, $10, $2
        nop
        nop
        bne $16, $10, skip8
        nop
        addiu $2, $2, 1554
skip8:  sw $2, 32($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $25, 1977109191
        addiu $20, $25, 0
        nop
        nop
        addiu $20, $25, 9834
        beq $25, $20, skip9
        nop
        nor $2, $2, $25
skip9:  sw $2, 36($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $24, -1206068049
        addiu $13, $24, 0
        nop
        addiu $13, $24, 28753
        nop
        beq $24, $13, skip10
        nop
        nor $2, $2, $24
skip10: sw $2, 40($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $21, 2080420061
        addiu $7, $21, 0
        srl $7, $21, 1
        nop
        nop
        beq $21, $7, skip11
        nop
        nor $2, $2, $21
skip11: sw $2, 44($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $24, -791642528
        addiu $22, $24, 0
        nop
        nop
        xori $22, $24, 6438
        bne $24, $22, skip12
        nop
        nor $2, $2, $24
skip12: sw $2, 48($0)
        mflo $2
        addu $2, $2, $4
        multu $2, $3
        li $8, -911568785
        addiu $22, $8, 0
        nop
        addiu $22, $8, 17950
        nop
        beq $8, $22, skip13
        nop
        nor $2, $2, $8
```

```
skip13:  sw $2, 52($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $21, 426530782
    addiu $8, $21, 6622
    ori $8, $21, 23640
    nop
    nop
    bne $21, $8, skip14
    nop
    nor $2, $2, $21
skip14:  sw $2, 56($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $18, 704083205
    addiu $19, $18, 28241
    nop
    nop
    sll $19, $18, 14
    beq $18, $19, skip15
    nop
    nor $2, $2, $18
skip15:  sw $2, 60($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $15, 1322704609
    addiu $11, $15, 0
    nop
    addiu $11, $15, 25990
    nop
    beq $15, $11, skip16
    nop
    nor $2, $2, $15
skip16:  sw $2, 64($0)
    mflo $2
    addu $2, $2, $4
    multu $2, $3
    li $29, -1891138629
    addiu $25, $29, 1450
    sll $25, $29, 26
    nop
    nop
    bne $29, $25, skip17
    nop
    nor $2, $2, $29
skip17:  sw $2, 68($0)
    mflo $2
    addu $2, $2, $4
    addu $28, $2, $8
    lui $8, 64451
    lui $28, 64451
    nop
beg18:   beq $8, $28, skip18
    addu $8, $0, $0
    beq $0, $0, beg18
    subu $28, $28, $28
skip18:  sw $28, 72($0)
    lui $8, 64451
    nop
    lui $28, 19666
beg19:   bne $8, $28, skip19
    addu $28, $0, $0
    beq $1, $1, beg19
    addu $8, $28, $2
skip19:  sw $8, 76($0)
    li $30, -1602503051
    mthi $30
    li $30, -1602503051
    li $23, 523843457
    nop
    nop
    mfhi $23
    beq $30, $23, skip20
    nop
    sw $30, 80($0)
skip20:  li $13, -1153218862
    mtlo $13
    li $13, -1153218862
    li $7, 638425955
    nop
    mflo $7
    nop
    bne $13, $7, skip21
    nop
    sw $13, 84($0)
```

六、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？

因为乘法运算和除法运算在部件进行工作时需要耗费比正常 ALU 更多的时间，如果直接把乘除法部件整合进 ALU 中，会大大增加整个 ALU 的组合逻辑运算时间，从而导致整个 CPU 的时钟周期增长。

对于乘除法运算而言，乘法的结果是 64 位，除法则有 32 位的余数和 32 位的商，无法在一条指令中将其写入普通的寄存器堆。所以需要独立的 HI、LO 寄存器，来保存乘除法的运算结果。

2、参照你对延迟槽的理解，试解释“乘除槽”。

由于乘执行乘法的时间为 5 个 cycle(包含写入内部的 HI 和 LO 寄存器)，执行除法的时间为 10 个 cycle，所以在乘除法指令执行的同时，其他乘除法指令不能再次进入乘除法部件。而在乘除法指令执行的过程中，其他的不需要乘除法部件的指令可以继续执行，所以为“乘除槽”，这也是需要有单独的乘除法部件而不是整合进 ALU 的好处。

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?

lb 等指令使用的数据扩展模块是组合逻辑，如果设计在 DM 之后，会导致 M 级所需要的时间增加，进而增大时钟周期。而在 W 级，它的组合逻辑电路很少，空闲时间较多，所以加入数据扩展模块不会增大时钟周期。

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）

在储存储存一个字符串的时候，因为字符串中的每个字符的大小只有 1 个字节，在按字节访问内存时，可以在满足需求的前提下大量节省空间，提高内存的利用率。而按字访问内存时，会造成大量的内存空闲，导致浪费，同时访问内存需要考虑内存地址与 4 的倍数关系，增加了运算操作。

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？

CPU 设计风格：

我在 P5 时使用的是规划者型的设计风格,但到了 P6 我改为了今年的新方法，采用暴力转发、随时转发，在保证后续指令需要使用值时，前序指令能及时将正确的运算结果转发给后续指令，不考虑指令的具体类型，只考虑寄存器的地址与写使能。

对抗复杂性：

1、在 Tuse 和 Tnew 的生成中，将指令归类为 cal_r, cal_i, load, store, branch, j, jr, jal, jalr 这几种，并建立 ControlHazards 模块，统一生成 Tuse 和 Tnew。

2、规范化命名，同时规定定义变量的位置，使变量能够被准确查找和修改。

3、预留空闲的端口，在需要增加输入/输出时方便操作。

6、你对流水线 CPU 设计风格有何见解？

流水线 CPU 的设计风格应该足够的直观易懂，方便后期纠错与修改，能够准确定位出错位置，简化增加指令的修改步骤。所有功能电路尽量模块化，命名与引用规范化，便于用工程化方法解决问题。

7、在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(非常重要)

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-M-RS	R	MEM	rs	addu $3,$1,$2 addu $5,$3,$4
2	R-M-RT	R	MEM	rt	addu $3,$1,$2 addu $5,$4,$3

3	R-W-RS	R	WB	rs	addu $3,$1,$2
					nop
					addu $5,$3,$4

4	R-W-RT	R	WB	rt	addu $3,$1,$2
					nop
					addu $5,$4,$3

5	I-M-RS	I	MEM	rs	ori $1,$0,20 addu $3,$1,$2
6	I-M-RT	I	MEM	rt	ori $1,$0,20 addu $3,$2,$1
7	I-W-RS	I	WB	rs	ori $1,$0,20 nop addu $3,$2,$2
8	I-W-RT	I	WB	rt	ori $1,$0,20 nop addu $3,$2,$3
9	lui-M-RS	lui	MEM	rs	lui $1,20 addu $3,$1,$2
10	lui-M-RT	lui	MEM	rt	lui $1,20 addu $3,$1,$2
11	lui-W-RS	lui	WB	rs	lui $1,20 nop addu $3,$1,$2
12	lui-W-RT	lui	WB	rt	lui $1,20 nop addu $3,$2,$1
13	L-M-RS	load	MEM	rs	lw $1,0($2) addu $3,$1,$2
14	L-M-RT	load	MEM	rt	lw $1,0($2) addu $3,$2,$1

15	L-W-RS	load	WB	rs	lw $1,0($2)
					nop
					addu $3,$1,$2

16	L-W-RT	load	WB	rt	lw $1,0($2)
					nop
					addu $3,$2,$1

17	J-M-RS	jal	MEM	rs	jal loop addu $2,$31,$1
18	J-M-RT	jal	MEM	rt	jal loop addu $2,$1,$31
19	J-W-RS	jal	WB	rs	jal loop nop addu $2,$31,$1
20	J-W-RT	jal	WB	rt	jal loop nop addu $2,$1,$31
21	R-E-RS	R	EXE	rs	addu $3,$1,$2 jr $3
22	R-M-RS	R	MEM	rs	addu $3,$1,$2 nop jr $3
23	R-W-RS	R	WB	rs	addu $3,$1,$2 nop nop jr $3
24	I-E-RS	I	EXE	rs	ori $1,$0,20 jr $1
25	I-M-RS	I	MEM	rs	ori $1,$0,20 nop jr $1
26	I-W-RS	I	WB	rs	ori $1,$0,20 nop nop jr $1
27	lui-E-RS	lui	EXE	rs	lui $1,20 jr $1
28	lui-M-RS	lui	MEM	rs	lui $1,20 nop jr $1

29	lui-W-RS	lui	WB	rs	lui $1,20 nop nop jr $1
30	L-E-RS	load	EXE	rs	lw $1,0($2) jr $1
31	L-M-RS	load	MEM	rs	lw $1,0($2) noo jr $1
32	L-W-RS	load	WB	rs	lw $1,0($2) noo nop jr $1
33	MD	MF	EXE	hi/lo	div $1,$2 mflo $3 mfhi $4