P7 设计文档

——MIPS 微体系

一、指令集

add	addu	sub	subu	sll	srl	sra	sllv	srlv	srav
and	or	xor	nor	slt	sltu	addi	addiu	andi	ori
xori	lui	slti	sltiu	beq	bne	blez	bgtz	bltz	bgez
j	jal	jalr	jr	lb	lbu	lh	lhu	lw	sb
sh	sw	mult	multu	div	divu	mfhi	mflo	mthi	mtlo
mtc0	mfc0	eret							

二、模块规格

1. PC：指令地址寄存器（F 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
en		I	写使能信号
PCI	[31:0]	I	下一条指令的地址
PC	[31:0]	O	当前指令的地址
exccode_PC	[4:0]	O	PC 部件的异常编码

模块功能说明如下：

序号	功能名	描述
1	读指令地址	通过 PC 端口输出当前指令地址
2	写指令地址	时钟上升沿到来时若 en 信号为 1，则更新指令地址
3	同步复位	时钟上升沿到来时若 reset 信号为 1，则复位指令地址至初始状态 0x00003000
4	输出异常编码	指令地址超出范围或未字对齐时输出相应异常编码

2. IM：指令存储器（F 级部件）

模块端口定义如下：

信号名	位数	方向	描述
IMI	[31:0]	I	当前指令地址信号
IM	[31:0]	O	当前指令信号

模块功能说明如下：

序号	功能名	描述
1	读指令	读出指令存储器中 IMI 地址对应的指令并通过 IM 端口输出
2	初始化写指令	初始时向指令存储器中读入所有指令

3. GRF： 寄存器堆（D 级部件，W 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
PC	[31:0]	I	当前指令的地址信号
A1	[4:0]	I	第一个读寄存器地址输入信号
A2	[4:0]	I	第二个读寄存器地址输入信号
A3	[4:0]	I	写寄存器地址输入信号
WD	[31:0]	I	写入数据信号
WE		I	写使能信号
RD1	[31:0]	O	A1 所对应的寄存器的数据信号
RD2	[31:0]	O	A2 所对应的寄存器的数据信号

模块功能说明如下：

序号	功能名	描述
1	读数据	读出 A1，A2 地址对应寄存器数据并通过 RD1，RD2 端口输出
2	写数据	时钟上升沿到来时，若 WE 信号为 1，则向 A3 地址对应寄存器写入数据 WD（0 号寄存器不能被写入）并输出一条与 PC 有关的信息
3	同步复位	时钟上升沿到来时，若 reset 信号为 1，则复位所有寄存器至初始状态 0x00000000

4. EXT：立即数拓展模块（D 级部件）

模块端口定义如下：

信号名	位数	方向	描述
EXTIMM	[15:0]	I	参与拓展的立即数信号
EXTOp	[1:0]	I	拓展方式的选择信号
EXT	[31:0]	O	拓展完成后的数据信号

模块功能说明如下：

序号	功能名	描述
1	符号拓展	若 EXTOp==00，将立即数加载至输出信号低位并用其最高位填充输出信号的高 16 位
2	零拓展	若 EXTOp==01，将立即数加载至输出信号低位并用 0 填充输出信号的高 16 位
3	低位零拓展	若 EXTOp==10，将立即数加载至输出信号高位并用 0 填充输出信号的低 16 位

5. NPC：指令地址计算器（D 级部件）

模块端口定义如下：

信号名	位数	方向	描述
ADD4	[31:0]	I	PC+4 信号
PC4_D	[31:0]	I	F/D 级流水线寄存器中的 PC4 信号
imm_index	[25:0]	I	计算跳转地址的 imm/index 数据信号
M_RD1_D	[31:0]	I	D 级读取 rs 寄存器时的转发信号
CMP		I	跳转时特殊条件的判断信号
Is_B		I	b 类指令的判断信号（beq/bne/blez/bgtz/bltz/bgez）
Is_J		I	j 类指令的判断信号（j / jal）
PCOp	[1:0]	I	计算下一条指令地址的选择信号
IntReq		I	中断请求信号
EPC	[31:0]	I	CP0 中 EPC 信号
TURE_NPC	[31:0]	O	下一条指令地址

模块功能说明如下：

序号	功能名	描述
1	计算指令地址	若 TntReq 有效，则输出 0x00004180 若 PCOp==00，输出 PC+4 信号 若 PCOp==01，输出 M_RD1_D 信号 若 PCOp==10 且 is_B==1 且 CMP==1，则计算 PC4_D+imm 拓展并输出 若 PCOp==10 且 is_J==1，则计算 PC4_D 与 index 拓展信号并输出 若 PCOp==10 且不属于以上情况，则输出 PC4_D+4 信号 若 PCOp==11，则输出 EPC 信号

6. CMP：数据比较器（D 级部件）

模块端口定义如下：

信号名	位数	方向	描述
D1	[31:0]	I	参与比较的第一个数据信号
D2	[31:0]	I	参与比较的第二个数据信号
CMPOp	[2:0]	I	比较方式的选择信号
CMP		O	比较结果信号

模块功能说明如下：

序号	功能名	描述
1	比较相等	若 CMPOp==000，二数据信号相等，则输出 1，否则输出 0
2	比较不等	若 CMPOp==001，若二数据信号不等，则输出 1，否则输出 0
3	比较小于 0	若 CMPOp==010，若 D1 小于 0，则输出 1，否则输出 0
4	比较小于等于 0	若 CMPOp==011，若 D1 小于等于 0，则输出 1，否则输出 0
5	比较大于 0	若 CMPOp==100，若 D1 大于 0，则输出 1，否则输出 0
6	比较大于等于 0	若 CMPOp==101，若 D1 大于等于 0，则输出 1，否则输出 0

7. ALU：计算模块（E 级部件）

模块端口定义如下：

信号名	位数	方向	描述
A	[31:0]	I	参与运算的第一个数据信号
B	[31:0]	I	参与运算的第二个数据信号
ALUOp	[3:0]	I	运算方式的选择信号
IR_E	[31:0]	I	E 级指令信号
ALU	[31:0]	O	运算结果的数据信号
exccode_ALU	[4:0]	O	ALU 部件的异常编码

模块功能说明如下：

序号	功能名	描述
1	按位与运算	若 ALUOp==0000，计算 A&B 并通过 ALU 端口输出
2	按位或运算	若 ALUOp==0001，计算 A|B 并通过 ALU 端口输出
3	支持溢出加法	若 ALUOp==0010，计算 A+B 并通过 ALU 端口输出
4	支持溢出减法	若 ALUOp==0011，计算 A-B 并通过 ALU 端口输出
5	逻辑左移	若 ALUOp==0100，计算 B 逻辑左移 A 并通过 ALU 端口输出
6	逻辑右移	若 ALUOp==0101，计算 B 逻辑右移 A 并通过 ALU 端口输出
7	算术右移	若 ALUOp==0110，计算 B 算术右移 A 并通过 ALU 端口输出
8	异或	若 ALUOp==0111，计算 A xor B 并通过 ALU 端口输出
9	或非	若 ALUOp==1000，计算 A nor B 并通过 ALU 端口输出
10	小于置 1	若 ALUOp==1001，A < B 则输出 1，否则输出 0
11	无符号小于置 1	若 ALUOp==1010，A 无符号 < B 则输出 1，否则输出 0
12	输出异常编码	运算溢出时输出相应异常编码

8. XALU：乘除法计算模块（E 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
XALUA	[31:0]	I	参与运算的第一个数据信号
XALUB	[31:0]	I	参与运算的第二个数据信号
XALUOp	[2:0]	I	运算方式的选择信号
START		I	乘除法计算的开始信号
BUSY		O	正在运算乘除法的判断信号
HIGH	[31:0]	O	HI 寄存器的值
LOW	[31:0]	O	LO 寄存器的值

模块功能说明如下：

序号	功能名	描述
1	同步复位	时钟上升沿到来时，若 reset 信号为 1，则复位 HI 和 LO 至 0x00000000，复位 BUSY 至 0
2	符号乘法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==000，则计算 A*B 并在 5 个时钟周期后通过 HIGH 和 LOW 输出
3	无符号乘法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==001，则计算无符号 A*B 并在 5 个时钟周期后通过 HIGH 和 LOW 输出
4	符号除法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==010，则计算 A/B 并在 10 个时钟周期后通过 HIGH 和 LOW 输出
5	无符号除法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==011，则计算无符号 A/B 并在 10 个时钟周期后通过 HIGH 和 LOW 输出
6	写 HI 寄存器	时钟上升沿到来时，若 XALUOp==100，则将 XALUA 值写入 HI
8	写 LO 寄存器	时钟上升沿到来时，若 XALUOp==100，则将 XALUA 值写入 LO

9. DM：数据存储器（M 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
PC	[31:0]	I	当前指令的地址信号
DMA	[31:0]	I	存取的地址信号
DMD	[31:0]	I	存取的数据信号
DMWE		I	存数据使能信号
DMOp	[1:0]	I	存数据方式的选择信号
IR_M	[31:0]	I	M 级指令信号
DM	[31:0]	O	取出的数据信号
exccode_DM	[4:0]	O	DM 部件的异常编码

模块功能说明如下：

序号	功能名	描述
1	存数据	时钟上升沿到来时，若 DMWE 信号为 1，则根据 DMOp 的值向数据存储器 DMA 对应地址的某一段中写入 DMD 数据并输出一条与 PC 有关的信息
2	取数据	将数据存储器 DMA 地址对应的数据通过 DM 端口输出
3	同步复位	时钟上升沿到来时，若 reset 信号为 1，则复位数据存储器至初始状态 0x00000000
4	输出异常编码	数据地址不对齐或超出范围时输出相应异常编码

10. DMEXT：数据存储器拓展器（M 级部件）

模块端口定义如下：

信号名	位数	方向	描述
DM	[31:0]	I	所需要拓展的数据信号
DMA	[31:0]	I	读取数据存储器的地址信号
DMEXTOp	[2:0]	I	拓展方式选择信号
DMEXT	[31:0]	O	拓展后的数据信号

模块功能说明如下：

序号	功能名	描述
1	拓展数据	根据 DMA 末两位值与 DMEXTOp 值对 DM 数据进行拓展后通过 DMEXT 端口输出

11. CP0：协处理器（M 级部件）（在下文中断异常处理一节中叙述）

三、流水线寄存器规格

1. FDreg：F/D 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	PC	IR_D
reset	IM	PC_D
en	exccode_PC	PC4_D
		PC8_D
		exccode_D

2. DEreg：D/E 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	IR_D	IR_E
reset	PC_D	PC_E
en	PC4_D	PC4_E
	PC8_D	PC8_E
	RD1	RD1_E
	RD2	RD2_E
	EXT	EXT_E
	GRFWE	GRFWE_E
	Tnew	Tnew_E
	exccode_D	exccode_E
	exccode_controller	

3. EMreg：E/M 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	IR_E	IR_M
reset	PC_E	PC_M
en	PC4_E	PC4_M
	PC8_E	PC8_M
	ALU	ALU_M
	RD2_E	RD2_M
	GRFWE_E	GRFWE_M
	Tnew_E	Tnew_M
	exccode_E	exccode_M
	exccode_ALU	

4. MWreg：M/W 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	IR_M	IR_W
reset	PC_M	PC_W
en	PC4_M	PC4_W
	PC8_M	PC8_W
	ALU_M	ALU_W
	DM	DM_W
	GRFWE_M	GRFWE_W

四、控制器设计

1. 控制信号说明

	0	1	2	3	4	5	6	7	8	9	10
GRFA3_MUXOp	rt	rd	31								
ALUB_MUXOp	RD2	EXT									
GRFWE	不写寄存器	写寄存器									
DMWE	不写DM	写DM									
GRFWD_MUXOp	ALU	DM	PC8	HI	LO						
EXTOp	符号拓展	零拓展	加载高位								
is_B		用imm算pc									
ALUOp	&	|	+	-	逻辑左移	逻辑右移	算术右移	xor	nor	小于置1	小于u置1
is_J	用index算pc										
PCOp	ADD4	RD1	NPC	EPC							
Tuse_rs	在npc用	在alu用	在dm用								
Tuse_rt	在npc用	在alu用	在dm用								
Tnew		在npc产生	在alu产生	在dm产生							
A3_Op	rd	rt	31								
WD_Op	pc8	alu	dm	HI	LO						
ALUA_MUXOp	RD1	shamt									
CMPOp	==	!=	<0	<=0	>0	>=0					
DMOp	word	halfword	byte								
DMEXTOp	word	halfword	halfwordu	byte	byteu						
XALUOp	*	*u	/	/u	mthi	mtlo					
START	用XALU计算										
CP0WE	不写CP0	写CP0									
TTDM_MUXOp	TDM	Dout									

2. 指令与控制信号真值表

	add	addu	sub	subu	sll	srl	sra	sllv	srlv	srav	and	or	xor	nor	slt	sltu
	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
	100000	100001	100010	100011	000000	000010	000011	000100	000110	000111	100100	100101	100110	100111	101010	101011
GRFA3_MUXOp	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ALUB_MUXOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRFWE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DMWE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRFWD_MUXOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EXTOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
is_B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ALUOp	2	2	3	3	4	5	6	4	5	6	0	1	7	8	9	10
is_J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PCOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tuse_rs	1	1	1	1	5	5	5	1	1	1	1	1	1	1	1	1
Tuse_rt	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Tnew	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
A3_Op	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WD_Op	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ALUA_MUXOp	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
DMOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DMEXTOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CPOWE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TTDM_MUXOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	addi	addiu	andi	ori	xori	lui	slti	sltiu		lb	lbu	lh	lhu	lw
	001000	001001	001100	001101	001110	001111	001010	001011		100000	100100	100001	100101	100011
GRFA3_MUXOp	0	0	0	0	0	0	0	0		0	0	0	0	0
ALUB_MUXOp	1	1	1	1	1	1	1	1		1	1	1	1	1
GRFWE	1	1	1	1	1	1	1	1		1	1	1	1	1
DMWE	0	0	0	0	0	0	0	0		0	0	0	0	0
GRFWD_MUXOp	0	0	0	0	0	0	0	0		1	1	1	1	1
EXTOp	0	0	1	1	1	2	0	0		0	0	0	0	0
is_B	0	0	0	0	0	0	0	0		0	0	0	0	0
ALUOp	2	2	0	1	7	2	9	10		2	2	2	2	2
is_J	0	0	0	0	0	0	0	0		0	0	0	0	0
PCOp	0	0	0	0	0	0	0	0		0	0	0	0	0
Tuse_rs	1	1	1	1	1	1	1	1		1	1	1	1	1
Tuse_rt	5	5	5	5	5	5	5	5		5	5	5	5	5
Tnew	2	2	2	2	2	2	2	2		3	3	3	3	3
A3_Op	1	1	1	1	1	1	1	1		1	1	1	1	1
WD_Op	1	1	1	1	1	1	1	1		2	2	2	2	2
ALUA_MUXOp	0	0	0	0	0	0	0	0		0	0	0	0	0
DMOp	0	0	0	0	0	0	0	0		0	0	0	0	0
DMEXTOp	0	0	0	0	0	0	0	0		3	4	1	2	0
CPOWE	0	0	0	0	0	0	0	0		0	0	0	0	0
TTDM_MUXOp	0	0	0	0	0	0	0	0		0	0	0	0	0

	beq	bne	blez	bgtz	bltz	bgez	j	jal	jalr	jr		sb	sh	sw
	000100	000101	000110	000111	000001	000001	000010	000011	000000	000000		101000	101001	101011
rt					00000	00001			001001	001000				
GRFA3_MUXOp	0	0	0	0	0	0	0	2	1	0		0	0	0
ALUB_MUXOp	0	0	0	0	0	0	0	0	0	0		1	1	1
GRFWE	0	0	0	0	0	0	0	1	1	0		0	0	0
DMWE	0	0	0	0	0	0	0	0	0	0		1	1	1
GRFWD_MUXOp	0	0	0	0	0	0	0	2	2	0		0	0	0
EXTOp	0	0	0	0	0	0	0	0	0	0		0	0	0
is_B	1	1	1	1	1	1	0	0	0	0		0	0	0
ALUOp	0	0	0	0	0	0	0	0	0	0		2	2	2
is_J	0	0	0	0	0	0	1	1	0	0		0	0	0
PCOp	2	2	2	2	2	2	2	2	1	1		0	0	0
Tuse_rs	0	0	0	0	0	0	5	5	0	0		1	1	1
Tuse_rt	0	0	5	5	5	5	5	5	5	5		2	2	2
Tnew	0	0	0	0	0	0	0	0	0	0		0	0	0
A3_Op	0	0	0	0	0	0	0	2	0	0		0	0	0
WD_Op	0	0	0	0	0	0	0	0	0	0		0	0	0
ALUA_MUXOp	0	0	0	0	0	0	0	0	0	0		0	0	0
CMPOp	0	1	3	4	2	5	0	0	0	0		0	0	0
DMOp	0	0	0	0	0	0	0	0	0	0		2	1	0
DMEXTOp	0	0	0	0	0	0	0	0	0	0		0	0	0
CPOWE	0	0	0	0	0	0	0	0	0	0		0	0	0
TTDM_MUXOp	0	0	0	0	0	0	0	0	0	0		0	0	0

	mult	multu	div	divu		mfhi	mflo	mthi	mtlo		mtc0	mfc0	eret
	000000	000000	000000	000000		000000	000000	000000	000000				
	011000	011001	011010	011011		010000	010010	010001	010011				
GRFA3_MUXOp	0	0	0	0		1	1	0	0		0	0	0
ALUB_MUXOp	0	0	0	0		0	0	0	0		0	0	0
GRFWE	0	0	0	0		1	1	0	0		0	1	0
DMWE	0	0	0	0		0	0	0	0		0	0	0
GRFWD_MUXOp	0	0	0	0		3	4	0	0		0	1	0
EXTOp	0	0	0	0		0	0	0	0		0	0	0
is_B	0	0	0	0		0	0	0	0		0	0	0
ALUOp	0	0	0	0		0	0	0	0		0	0	0
is_J	0	0	0	0		0	0	0	0		0	0	0
PCOp	0	0	0	0		0	0	0	0		0	0	3
Tuse_rs	1	1	1	1		0	0	1	1		5	5	5
Tuse_rt	1	1	1	1		0	0	5	5		2	5	5
Tnew	2	2	2	2		3	3	2	2		3	3	0
A3_Op	0	0	0	0		0	0	0	0		0	1	0
WD_Op	0	0	0	0		3	4	0	0		0	2	0
ALUA_MUXop	0	0	0	0		0	0	0	0		0	0	0
CMPOp	0	0	0	0		0	0	0	0		0	0	0
DMOp	0	0	0	0		0	0	0	0		0	0	0
DMEXTOp	0	0	0	0		0	0	0	0		0	0	0
XALUOp	0	1	2	3		0	0	4	5		0	0	0
START	1	1	1	1		0	0	0	0		0	0	0
CPOWE	0	0	0	0		0	0	0	0		1	0	0
TTDM_MUXOp	0	0	0	0		0	0	0	0		0	1	0

五、冒险处理策略

在 P6 中共有 50 条指令，仍使用 P5 中所采取的分类表格法将十分繁杂且易于出错，因此，本设计采取暴力转发的方法。其主要实现方式为，在每一级流水线寄存器中都存入当前指令的 Tnew（距离产生结果所剩的时钟周期数）以及 A3（所将要写入的 5 位寄存器地址）和 WD（所将要写入的数据），同时 D 级控制器产生 D 级指令的 Tuse_rs（距需要使用 RD1 所剩的时钟周期数）和 Tuse_rt（距需要使用 RD2 所剩的时钟周期数）。每一次时钟上升沿到来时，若检测到 D 级指令需要用到的 RD1 或 RD2 仍未产生，则产生一次暂停，若检测到其他已产生结果的数据冒险，则进行转发。这种设计保证了每一条指令所使用的值均是正确的。此外，乘除法的暂停也由 BUSY 信号决定。

在 P7 中新加入了 eret 指令，为了防止其后的非 nop 指令生效，也需要暂停。

有关模块：

1. Analysis_MUX（A3 及 WD 分析模块）

功能：根据每一级控制器产生的 A3_MUX 和 WD_MUX 信号产生每一级指令的 A3 与 WD

2. StopUnit（暂停分析与产生模块）

功能：检测到 D 级指令所需数据不能及时得出时进行一次暂停，D 级为乘除法相关指令且 XALU 忙碌时进行一次暂停

3. ForwardUnit（转发来源分析与产生模块）

功能：在 D 级 rs、D 级 rt、E 级 rs、E 级 rt、M 级 rt 五个位点检测到数据冒险时根据各个流水线寄存器的 WD 和 A3 值进行转发

六、桥与 IO 设计

桥的端口定义如下：

信号名	位数	方向	描述
PrAddr	[31:0]	I	CPU 发出的数据地址信号
PrWD	[31:0]	I	CPU 发出的写入数据信号
PrWE		I	CPU 发出的数据写使能信号
DEV0RD	[31:0]	I	Timer0 发出的读出数据信号
DEV1RD	[31:0]	I	Timer1 发出的读出数据信号
PrRD	[31:0]	O	外设总读出数据信号
DEVAddr	[31:0]	O	外设数据地址信号
DEVWD	[31:0]	O	外设写入数据信号
DEV0WE		O	Timer0 写使能信号
DEV1WE		O	Timer0 写使能信号

桥的功能说明如下：

序号	功能名	描述
1	译码	将 CPU 发出的地址、写使能、数据信号转换成相应外设的地址、写使能、数据信号，将外设发出的读数据信号转换成总读数据信号

七、CP0 设计

在本设计中，CP0 实现的内部寄存器有 12 号 SR 读写寄存器（im[15:10], exl, ie）、13 号 Cause 只读寄存器（bd, hwint_pend[15:10], exccode[6:2]）、14 号 EPC 读写寄存器、15 号 PRId 只读寄存器。

注：在本设计中，CP0 处于 M 级。

CP0 端口定义如下：

信号名	位数	方向	描述
A1	[4:0]	I	读 CP0 寄存器编号
A2	[4:0]	I	写 CP0 寄存器编号
Din	[31:0]	I	写入数据信号
PC	[31:2]	I	指令地址信号
ExcCode	[6:2]	I	当前指令异常类型信号
HWInt	[5:0]	I	外部中断信号
WE		I	写使能信号
EXLSet		I	用于置位 SR 的 EXL 位
ExlClr		I	用于清除 SR 的 EXL 位
clk		I	内置时钟信号
reset		I	同步复位信号
IR_W	[31:0]	I	流水线 W 级指令信号
IntReq		O	中断请求信号
EPC	[31:2]	O	异常处理结束后返回的指令地址信号
Dout	[31:0]	O	读出数据信号

CP0 功能说明如下：

序号	功能名	描述
1	写数据	WE 有效时向 A2 寄存器中写入 Din 值
2	读数据	将 A1 寄存器值通过 Dout 端口输出
3	同步复位	reset 信号有效时复位所有寄存器至 0x00000000
4	锁存外部中断	将 HWInt 信号不断记录至 hwint_pend 位
5	进入内核态	出现外部中断或 M 级指令异常时，进入内核态置 exl 为 1，记录 EPC、bd、exccode
6	从内核态返回	EXLClr 有效时从内核态返回

八、异常与中断设计

1. 异常行为规范

本设计中，对所有异常都遵循精确异常的处理规则：即受害指令的前序指令的期望执行结果都应执行，并且受害指令及其后序指令的执行效果不影响异常处理程序返回后所执行指令的执行效果，具体异常分类与 exccode 编码如下：

ExcCode	名称与助记符	指令或指令类型	描述与备注	备注
4	AdEL（取指异常）	所有指令	PC地址未4字节对齐	1. 测试时不会出现跳转到未加载指令的位置。2. 发生取指异常后视为nop直至提交至CP0.
		所有指令	PC地址超出0x3000-0x4ffc范围	
	AdEL（取数异常）	lw	取数地址未4字节对齐	
		lh、lhu	取数地址未2字节对齐	
		lh、lhu、lb、lbu	取Timer寄存器的值	
		Load类	计算地址加法溢出	
		Load类	取数地址超出DM、Timer0、Timer1的范围	
5	AdEs（存数异常）	sw	存数地址未4字节对齐	
		sh	存数地址未2字节对齐	
		sh、sb	存Timer寄存器的值	
		Store类	计算地址溢出	
		Store类	向计时器的Count寄存器存值	
		Store类	存数地址超出DM、Timer0、Timer1的范围	
10	RI（未知指令）	-	未知的指令码	1. 课上测试的未知指令的测试点中，非法指令的Opcode和Func码组合一定没有在正确指令集中出现。2. 发生RI异常后视为nop直至提交至CP0.
12	Ov（溢出异常）	add、addi、sub	算术溢出	store与load类算址溢出按照AdEL或AdES

注：分支跳转指令无论跳转与否，延迟槽指令为受害指令时 BD 均需要置位。

如果 jr 跳转到一个不对齐的地址，那么 EPC 的值应该为该地址，而非 jr 指令的 PC 值。

2. 异常判断方式

在本设计中，PC、D 级 controller、ALU、DM 部件会判断当前指令的异常类型，记录其 exccode（无异常则即为 0）并随流水线向下传递，传递时流水线寄存器优先取前一级流水线寄存器记录的 exccode 值，若为 0 则取该级功能部件发出的 exccode 值，并在 M 级交由 CP0 处理。

3. 中断行为规范

本实验不要求在中断发生时指定具体受害指令，但需要保证中断的处理是精确的——内核态前后每条指令的执行效果不变。CPU 共需接收来自两个 Timer 模拟的外部中断和一个来自 mips 模块外的外部中断信号。

4. 异常与中断处理

在 M 级 CP0 接收到外部中断或内部 M 级指令异常后，经内部逻辑处理，记录各类异常中断信息（exccode、bd、hwint_pend、epc）并发出 IntReq 信号，清除仍在流水线中的指令并进入内核态，屏蔽外部中断。当 M 级指令为 eret 时从内核态返回。

九、其他

地址范围声明如下：

	地址或地址范围	备注
数据存储器	0x0000_0000至0x0000_2FFF	
指令存储器	0x0000_3000至0x0000_4FFF	
PC初始值	0x0000_3000	
Exception Handler入口地址	0x0000_4180	
定时器寄存器地址	0x0000_7F00至0x0000_7F0B 0x0000_7F10至0x0000_7F1B	定时器0的3个寄存器 定时器1的3个寄存器

序号	时间	内容	方式
1	12.5	修复了乘除法指令BUSY周期不正确的bug	将乘除法预设BUSY周期减小1
2	12.11	修复了mtc0 + eret没有暂停的bug	stop_unit加入eret暂停条件
3	12.11	修复了在由于暂停产生的空泡发生中断时EPC不正确的bug	E级寄存器清空时PC_E改为PC_D
4	12.11	修复了由于暂停产生空泡后不能正确判断是否是延迟槽指令的bug	在F级判断BD并随流水线传递
5	12.11	修复了DM中lbu和lb异常条件不正确的bug	修改了DM异常条件
6	12.12	修复了store类指令地址异常时仍会写入数据的bug	修改了DM中store类指令写入条件，修改了PrWE有效条件
7	12.12	修复了异常指令 + 乘除法指令仍会向hilo中写值或开始计算乘除法的bug	增加DXALU控制信号IntReq
8	12.12	修复了DMWE连接信号不正确的bug	将DMWE端口连接信号改为TDMWE
9	12.12	修复了在由于暂停产生的空泡发生中断时bd不正确的bug	E级寄存器清空时bd_E改为bd_D
10	12.15	修复了CP0中EPC在bd有效时不正确的bug	CP0中EPC条件中改为bd_M
11	12.15	修复了eret暂停时PC不正确加载的bug	FD级reset条件加入FDen
12	12.15	修复了乘除法指令会对异常时PC更新造成影响的bug	提高了PC中异常更新4180的优先级
13	12.18	修复了eret在M级时下中断后EPC更新不正确的bug	eret清除FD流水线寄存器时将PC_D改写为EPC
14			
15			
16			
17			
18			
19			
20			
21			

十、测试程序

命令行导出：

```
java -jar E:\Mars4_5.jar a db mc CompactDataAtZero dump 0x00004180-0x00004ffc HexText E:\code_handler.txt E:\source.asm
```

测试主要从以下四方面入手：CPU，异常，中断，IO

1. CPU 测试主要由 P6 的自动化测试完成，并手动构造关于 CP0 的三条指令的样例
2. 异常测试主要是手动构造遍历各种异常情况

例如下：

```
.ktext 0x4180
_entry:
```

```
    mfc0   $k0, $14
    mfc0   $k1, $13
    ori$k0, $0, 0x1000
    sw $sp, -4($k0)

    addiu  $k0, $k0, -256
    move   $sp, $k0

    j  _save_context
    nop

_main_handler:
    mfc0   $k0, $13
    ori    $k1, $0, 0x007c
    and$k0, $k1, $k0
    beq    $0, $k0, _restore_context
    nop
    mfc0   $k0, $14
    addu   $k0, $k0, 4
    mtc0   $k0, $14
    j  _restore_context
    nop

_restore:
    eret

_save_context:
    sw     $1, 4($sp)
        sw     $2, 8($sp)
        sw     $3, 12($sp)
        sw     $4, 16($sp)
        sw     $5, 20($sp)
        sw     $6, 24($sp)
        sw     $7, 28($sp)
        sw     $8, 32($sp)
        sw     $9, 36($sp)
        sw     $10, 40($sp)
        sw     $11, 44($sp)
        sw     $12, 48($sp)
        sw     $13, 52($sp)
        sw     $14, 56($sp)
        sw     $15, 60($sp)
        sw     $16, 64($sp)
        sw     $17, 68($sp)
        sw     $18, 72($sp)
        sw     $19, 76($sp)
        sw     $20, 80($sp)
        sw     $21, 84($sp)
        sw     $22, 88($sp)
        sw     $23, 92($sp)
        sw     $24, 96($sp)
        sw     $25, 100($sp)
        sw     $26, 104($sp)
        sw     $27, 108($sp)
        sw     $28, 112($sp)
        sw     $29, 116($sp)
        sw     $30, 120($sp)
        sw     $31, 124($sp)
    mfhi   $k0
    sw $k0, 128($sp)
    mflo   $k0
    sw $k0, 132($sp)
    j  _main_handler
    nop



_restore_context:
    lw $1, 4($sp)
```

```
        lw     $2, 8($sp)
        lw     $3, 12($sp)
        lw     $4, 16($sp)
        lw     $5, 20($sp)
        lw     $6, 24($sp)
        lw     $7, 28($sp)
        lw     $8, 32($sp)
        lw     $9, 36($sp)
        lw     $10, 40($sp)
        lw     $11, 44($sp)
        lw     $12, 48($sp)
        lw     $13, 52($sp)
        lw     $14, 56($sp)
        lw     $15, 60($sp)
        lw     $16, 64($sp)
        lw     $17, 68($sp)
        lw     $18, 72($sp)
        lw     $19, 76($sp)
        lw     $20, 80($sp)
        lw     $21, 84($sp)
        lw     $22, 88($sp)
        lw     $23, 92($sp)
        lw     $24, 96($sp)
        lw     $25, 100($sp)
        lw     $26, 104($sp)
        lw     $27, 108($sp)
        lw     $28, 112($sp)
        lw     $29, 116($sp)
        lw     $30, 120($sp)
        lw     $31, 124($sp)
    lw $k0, 128($sp)
    mthi   $k0
    lw $k0, 132($sp)
    mtlo   $k0
        j   _restore
    nop



.text
    ori    $2, $0, 0x1001
        mtc0   $2, $12
    ori $28, $0, 0x0000
    ori $29, $0, 0x0000
    lui $8, 0x7fff
    lui $9, 0x7fff
    add $10, $8, $9
    or  $10, $8, $9

end:
    beq $0, $0, end
    nop
```

3. 中断测试使用 P6 自动生成的 text 代码并在 TB 中随机加入中断信号

   TB 例如下：

```
module mips_txt;

    // Inputs
    reg clk;
    reg reset;
    reg interrupt;

    // Outputs
    wire [31:0] addr;


    // Instantiate the Unit Under Test (UUT)
    mips uut (
```

```
        .clk(clk),
        .reset(reset),
        .addr(addr),
        .interrupt(interrupt)
    );

    parameter exception_pc = 32'h00003018;
    integer exception_count;
    integer interrupt_counter;
    integer needInterrupt;

    initial begin
        exception_count = 0;
        interrupt = 0;
        needInterrupt = 0;
        interrupt_counter = 0;
        // Initialize Inputs
        clk = 0;
        reset = 1;
        #20 reset = 0;
        // Wait 100 ns for global reset to finish
        // Add stimulus here

    end
   always #2 clk = ~clk;

   always @(negedge clk) begin
      if (reset) begin
        interrupt_counter = 0;
        needInterrupt = 0;
        interrupt = 0;
      end else begin
        if (interrupt) begin
            if (interrupt_counter == 0) begin
                interrupt = 0;
            end else begin
                interrupt_counter = interrupt_counter - 1;
            end
        end else if (needInterrupt) begin
            needInterrupt = 0;
            interrupt = 1;
            interrupt_counter = 5;
        end else begin
            case (addr)
                exception_pc:
                    begin
                        if (exception_count == 0) begin
                            exception_count = 1;
                            interrupt = 1;
                            interrupt_counter = 5;
                        end
                    end
            endcase
        end
      end

   end

endmodule
```

4. I0 构造少许样例即可

思考题

1. 我们计组课程一本参考书目标题中有“硬件/软件接口”字样，那么到底什么是“硬件/软件接口”？

硬件/软件接口就是计算机中硬件与软件的交互手段。硬件接口既包括物理上的接口，也包括逻辑上的数据传送协议，软件接口通常是驱动程序。

2. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

DM 应该是与 CPU 直接相连的 cache，CPU 通过 cache 与主存交换数据。

3. BE 部件对所有的外设都是必要的吗？

不全是必要的，某些外设（如本实验中的 Timer）只支持按字读写，不支持按半字或字节读写，因此 BE 部件不必要。

4. 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

见《计时器说明文档》。

5. 请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能：

(1) 定时器在主程序中被初始化为模式 0；
(2) 定时器倒计时数至 0 产生中断；
(3) handler 设置使能 Enable 为 1 从而再次启动定时器的计数器，2 及 3 被无限重复；
(4) 主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。（注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行）

```
.ktext 0x4180
ori $t9,0x7f00
ori $t8,0x0009
sw $t8,0($t9)        ## ctrl[3:0] = 4'b1001
eret

.text
ori $t1,0x7f00
ori $t2,0x0009
sw $t2,0($t1)        ## ctrl[3:0] = 4'b1001
ori $t3,100
sw $t3,4($t1)        ## preset = 100
```

6. 请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

外设包括设备主体和接口控制器两部分，设备主体通过接口控制器与主机连接。鼠标和

键盘的输入信号相当于中断，当鼠标和键盘产生输入信号时，会产生一个中断然后中断例程会从端口读入数据到寄存器，CPU 接收到中断后进入中断处理程序获取鼠标和键盘的输入信息。