MIPS 微系统

设计文档

贾博驿

2023 年 12 月 3 日

设计实现指令

共 54 条

R 型指令：

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

I 型指令：

6	5	5	16
opcode	rs (base)	rt	imm16 (immediate, offset)

J 型指令：

6	26
opcode	instr_index

注：

1.以下的 RTL 除分支与跳转类指令均省略 PC←PC+4。

2.测试程序保证未知指令的测试用例中 opcode 和 funct 码的组合一定没有在 MIPS 基本指令集中出现，故没有对之前实现的指令进行删除。

算数与逻辑类

1. 加减法类

a. R 型

名称	funct	RTL

add	100000	temp ← (GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) + (GPR[rt]$_{31}$||GPR[rt]$_{31..0}$)
		if temp$_{32}$ ≠ temp$_{31}$ then
		SignalException(Ov)
		else
		GPR[rd] ← temp$_{31..0}$
		endif

addu	100001	GPR[rd] ← GPR[rs] + GPR[rt]

sub	100010	temp ← (GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) − (GPR[rt]$_{31}$||GPR[rt]$_{31..0}$)
		if temp$_{32}$ ≠ temp$_{31}$ then
		SignalException(Ov)
		else
		GPR[rd] ← temp$_{31..0}$
		endif

subu	100011	GPR[rd] ← GPR[rs] - GPR[rt]

b. I 型

名称	opcode	RTL

addi	001000	temp ← (GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) + sign_extend(immediate)
		if temp$_{32}$ ≠ temp$_{31}$ then
		SignalException(Ov)
		else
		GPR[rt] ← temp$_{31..0}$
		endif

addiu	001001	GPR[rt] ← GPR[rs] + sign_extend(immediate)

2. 逻辑类

a. R 型

名称	funct	RTL
and	100100	GPR[rd] ← GPR[rs] AND(bitwise) GPR[rt]
or	100101	GPR[rd] ← GPR[rs] OR(bitwise) GPR[rt]
xor	100110	GPR[rd] ← GPR[rs] XOR(bitwise) GPR[rt]
nor	100111	GPR[rd] ← GPR[rs] NOR(bitwise) GPR[rt]
sll	000000	GPR[rd] ← GPR[rt]$_{(31-shamt)..0}$ || 0^{shamt}
srl	000010	GPR[rd] ← 0^{shamt} || GPR[rt]$_{31..shamt}$
sra	000011	GPR[rd] ← (GPR[rt]$_{31}$)^{shamt} || GPR[rt]$_{31..shamt}$
sllv	000100	s ← GPR[rs]$_{4..0}$ GPR[rd] ← GPR[rt]$_{(31-s)..0}$ || 0^{s}
srlv	000110	s ← GPR[rs]$_{4..0}$ GPR[rd] ← 0^{s} || GPR[rt]$_{31..s}$
srav	000111	s ← GPR[rs]$_{4..0}$ GPR[rd] ← (GPR[rt]$_{31}$)^{s} || GPR[rt]$_{31..s}$

注：根据寄存器值移位指令只依照寄存器值的低五位移位，不受高位值影响。

b. I 型

名称	opcode	RTL
andi	001100	GPR[rt] ← GPR[rs] AND(bitwise) zero_extend(immediate)
ori	001101	GPR[rt] ← GPR[rs] OR(bitwise) zero_extend(immediate)
xori	001110	GPR[rt] ← GPR[rs] XOR(bitwise) zero_extend(immediate)

3. 乘除法类（R 型）

名称	funct	RTL
mult	011000	prod ← GPR[rs]$_{31..0}$ × GPR[rt]$_{31..0}$ LO ← prod$_{31..0}$ HI ← prod$_{63..32}$
multu	011001	prod ← (0 || GPR[rs]$_{31..0}$) × (0 || GPR[rt]$_{31..0}$) LO ← prod$_{31..0}$ HI ← prod$_{63..32}$
div	011010	q ← GPR[rs]$_{31..0}$ ÷ GPR[rt]$_{31..0}$ r ← GPR[rs]$_{31..0}$ % GPR[rt]$_{31..0}$ LO ← q HI ← r
divu	011011	q ← (0 || GPR[rs]$_{31..0}$) ÷ (0 || GPR[rt]$_{31..0}$) r ← (0 || GPR[rs]$_{31..0}$) % (0 || GPR[rt]$_{31..0}$) LO ← q$_{31..0}$ HI ← r$_{31..0}$
mfhi	010000	GPR[rd] ← HI
mthi	010001	HI ← GPR[rs]
mflo	010010	GPR[rd] ← LO
mtlo	010011	LO ← GPR[rs]

寄存器操作类

a. R 型

名称	funct	RTL
slt	101010	GPR[rd] ← 0$^{GPRLEN-1}$ || GPR[rs] < GPR[rt]
sltu	101011	GPR[rd] ← 0$^{GPRLEN-1}$ || (0 || GPR[rs]) < (0 || GPR[rt])

注：slt 类指令成立时置 1，**不成立时置 0**。

b. I 型

名称	opcode	RTL
slti	001010	GPR[rt] ← $0^{GPRLEN-1}$ || GPR[rs] < sign_extend(immediate)
sltiu	001011	GPR[rt] ← $0^{GPRLEN-1}$ || (0 || GPR[rs]) < (0 || **sign_extend**(immediate))
lui	001111	GPR[rt] ← immediate || 0^{16}

注：sltiu 对立即数的处理是先**有符号扩展**再进行**无符号比较**。

分支与跳转类

1. 分支类（I 型）

名称	opcode	rt	RTL
(all)			**I+1:** if condition then PC ← PC + sign_extend(offset || 0^2) endif
bltz	000001	00000	**I:** condition ← GPR[rs] < 0^{32}
bgez	000001	00001	**I:** condition ← GPR[rs] ≥ 0^{32}
beq	000100	rt	**I:** condition ← (GPR[rs] = GPR[rt])
bne	000101	rt	**I:** condition ← (GPR[rs] ≠ GPR[rt])
blez	000110	00000	**I:** condition ← GPR[rs] ≤ 0^{32}
bgtz	000111	00000	**I:** condition ← GPR[rs] > 0^{32}

注：流水线 CPU 分支指令不需要 PC + 4 后再加上 offset，直接在延迟槽指令的地址上加。

2. 跳转类

a. R 型

名称	funct	RTL
jr	001000	**I:** **I+1:** PC ← GPR[rs]
jalr	001001	**I:** GPR[rd] ← PC + 8 **I+1:** PC ← GPR[rs]

b. J 型

名称	opcode	RTL
j	000010	**I:** **I+1:** PC ← $PC_{31..28}$ || instr_index || 0^2
jal	000011	**I:** GPR[31] ← PC + 8 **I+1:** PC ← $PC_{31..28}$ || instr_index || 0^2

注：流水线 CPU 跳转指令由于延迟槽存在应该加上 PC + 8，而非单周期的 PC + 4。

系统桥操作类（I 型）

名称	opcode	RTL
(all)		addr ← sign_extend(offset) + GPR[base]
lb	100000	byte ← $addr_{1..0}$ GPR[rt] ← sign_extend(memory[addr]$_{7+8*byte..8*byte}$)

lh	100001	if $addr_0$ ≠ 0 then
		SignalException(AdEL)
		endif
		half ← $addr_1$

		GPR[rt] ← sign_extend(memory[addr]$_{15+8*half..8*half}$)

lw	100011	if addr$_{1..0} \neq 0_2$ then
		SignalException(AdEL)
		endif
		GPR[rt] ← memory[addr]

lbu	100100	byte ← addr$_{1..0}$ GPR[rt] ← zero_extend(memory[addr]$_{7+8*byte..8*byte}$)

lhu	100101	if addr$_0 \neq 0$ then
		SignalException(AdEL)
		endif
		half ← addr$_1$
		GPR[rt] ← zero_extend(memory[addr]$_{15+8*half..8*half}$)

sb	101000	memory[addr]$_{7..0}$ ← GPR[rt]$_{7..0}$

sh	101001	if addr$_0 \neq 0$ then
		SignalException(AdES)
		endif
		memory[addr]$_{15..0}$ ← GPR[rt]$_{15..0}$

sw	101011	if addr$_{1..0} \neq 0_2$ then
		SignalException(AdES)
		endif
		memory[addr]$_{31..0}$ ← GPR[rt]

注：

1.针对字节和半字操作时只修改对应字节和半字的值，**同一字的其他部分不变**。

2.计算地址时加法溢出、地址不在存取合法地址区段等情况也会引发对应的 AdE 异常，具体见中断与异常表。

异常处理类

mfc0（COP0 型）：GPR[rt] ← CPR[0,rd]

opcode	rs	rt	rd	0	sel
010000	00000			00000000	000

mtc0（COP0 型）：CPR[0, rd] ← GPR[rt]

opcode	rs	rt	rd	0	sel
010000	00100			00000000	000

eret（COP0 型）：SR$_{EXL}$ ← 0 PC ← EPC

opcode	rs	0	funct
010000	10000	000000000000000	011000

syscall（R 型）：SignalException(Syscall)

opcode	0	funct
000000	00000000000000000000	001100

注：

1.因为 P7 实现的 CP0 不涉及使用 sel 域进行同编号寄存器的选择功能，认为其固定为 000，若是非 000 值也不响应。

2.eret 指令判断组合为 opcode 与 funct，而不是 opcode 与 rs，**且 eret 指令没有延迟槽。**

转发与暂停设计

指令分类

类型名	信号名	数量	指令
算数逻辑寄存器	ALREG	16	_arith（算数）：add, addu, sub, subu _logic（逻辑）：and, or, xor, nor _shift（移位）：sll, srl, sra, sllv, srlv, srav _comp（比较）：slt, sltu
算数逻辑立即数	ALIMM	7	_sign（有符号拓展）：addi, addiu, slti, sltiu _zero（无符号拓展）：andi, ori, xori
乘除法	MLUTDIV	4	mult, multu, div, divu
乘除寄存器读取	MLFROM	2	mfhi, mflo
乘除寄存器写入	MLTO	2	mthi, mtlo
高位加载	EXTLUI	1	lui
两数比较分支	BRANCHE	2	beq, bne
与零比较分支	BRANCHZ	4	bltz, bgez, blez, bgtz
仅读取跳转	JUMPR	1	jr
仅写入跳转	JUMPW	1	jal
读取写入跳转	JUMPRW	1	jalr
系统桥读取	LOAD	5	lb, lh, lw, lbu, lhu
系统桥写入	STORE	3	sb, sh, sw
读取协处理器寄存器	MFCOP	1	mfc0
写入协处理器寄存器	MTCOP	1	mtc0

注：不包括 j、eret、syscall 指令，因为它们不读取通用寄存器也不写入通用寄存器。

Tuse 与 Tnew

信号名	Tuse_rs	Tuse_rt
ALREG	1	1
ALIMM	1	(3)
MLUTDIV	1	1
MLFROM	(3)	(3)
MLTO	1	(3)
EXTLUI	(3)	(3)
BRANCHE	0	0
BRANCHZ	0	(3)
JUMPR	0	(3)
JUMPW	(3)	(3)
JUMPRW	0	(3)
LOAD	1	(3)
STORE	1	2
MFCOP	(3)	(3)
MTCOP	(3)	2

GRF_WD_W_Sel	Tnew_E	Tnew_M	Tnew_W
00	1	0	0
00	1	0	0
(00)	(0)	(0)	(0)
00	1	0	0
(00)	(0)	(0)	(0)
10	0	0	0
(00)	(0)	(0)	(0)
(00)	(0)	(0)	(0)
(00)	(0)	(0)	(0)
11	0	0	0
11	0	0	0
01	2	1	0
(00)	(0)	(0)	(0)
01	2	1	0
(00)	(0)	(0)	(0)

注：

1.(3)表示该类指令不会读取 rs 或 rt 域翻译为寄存器编号后对应的寄存器，除 GRF 内部转发外不产生任何转发和暂停。

2.(00)和(0)表示表示该类指令 GRFWE 为 0，虽然使用 ALU 数据但无意义。

转发与暂停表

SPL_rs != 5'd0 && GRFWE_X && SPL_rs == GRF_A3_X									
流水段(X)	E				M (~ rs_E_premise)				W
GRF_WD_W_Sel_X(Tnew_X)	00(1)	01(2)	10(0)	11(0)	00(0)	01(1)	10(0)	11(0)	All
Tuse_rs == 0	S1	S2	F43	F44	F21	S3	F23	F24	F1
Tuse_rs == 1	F51	S4				F32			
SPL_rt != 5'd0 && GRFWE_X && SPL_rt == GRF_A3_X									
流水段(X)	E				M (~ rt_E_premise)				W
GRF_WD_W_Sel_X(Tnew_X)	00(1)	01(2)	10(0)	11(0)	00(0)	01(1)	10(0)	11(0)	All
Tuse_rt == 0	S1	S2	F43	F44	F21	S3	F23	F24	F1
Tuse_rt == 1	F51	S4				F32			
Tuse_rt == 2		F62							
ISMULTDIV & (MULT_Start | MULT_Busy)									
SMULTDIV									
CP0WE_X && rd_X == 5'd14 (EPC)									
流水段(X)	E				M				
/	FEPC_E				FEPC_M				

注：

1.优先级 **S > F4、F5、F6 > F2、F3。**

2.F2 和 F3 之间、F4、F5 和 F6 之间不会发生冲突，因为转发的是同一流水级的不同数据。

转发信号表

SPL_rs != 5'd0 && GRFWE_X && SPL_rs == GRF_A3_X			
类型	描述	FMUX_V1_D_Sel	FMUX_V1_E_Sel
S	暂停		
F51	V1_E 转发 E_RES_M		11
F43	V1_D 转发 ext32_E	110	
F44	V1_D 转发 pc8_E	111	
F32	V1_E 转发 M_RES_W		10
F21	V1_D 转发 E_RES_M	011	
F23	V1_D 转发 ext32_M	100	
F24	V1_D 转发 pc8_M	101	
F1	GRF 内部转发		
F0	不转发	000	00

SPL_rt != 5'd0 && GRFWE_X && SPL_rt == GRF_A3_X				
类型	描述	FMUX_V2_D_Sel	FMUX_V2_E_Sel	FMUX_V2_M_Sel
S	暂停			
F62	V2_M 转发 M_RES_W			1

F51	V2_E 转发 E_RES_M		11	
F43	V2_D 转发 ext32_E	110		
F44	V2_D 转发 pc8_E	111		
F32	V2_E 转发 M_RES_W		10	
F21	V2_D 转发 E_RES_M	011		
F23	V2_D 转发 ext32_M	100		
F24	V2_D 转发 pc8_M	101		
F1	GRF 内部转发			
0	不转发	000	00	0

注：未标注的信号取值与不转发相同。

CP0WE_X && rd_X == 5'd14 (EPC)		
类型	描述	FMUX_EPC_F
FEPC_E	EPC_E 转发 FMUX_V2_E	FMUX_V2_E
FEPC_M	CP0 内部转发	CP0_Q_EPC
0	不转发	CP0_Q_EPC

暂停变化表

I	I+1	I+2
S1	F21	
S2	S3	F1

I	I+1	I+2
S3	F1	
S4	F32	

暂停操作

1.数据寄存器

1)FR_D 暂停使能，保留取出指令的信息。

2)FR_E 清空数据，等价于将指令变成 nop。

2.异常寄存器

1)ER_D 暂停使能，保留取指过程信息。

2)ER_E 流水 ER_D 的数据，不清空数据，维护宏观 PC，确保在暂停过程中出现中断时 EPC 正常写入数据。

注：如果发生取指异常或者未知指令异常，指令会被强制变成 nop，不会引发转发或暂停，也不会在流水过程中引发其他的异常。

异常与中断设计

中断与异常表

ExcCode	助记符	描述	指令	流水级	发生情况
0	Int	外部中断	(Any)	M	设备中断请求
4	AdEL	取指异常	(Any)	F	PC 地址未字对齐或
			(Any)	F	PC 地址不在 0x3000-0x6ffc
		取数异常	LOAD	E	计算地址时加法溢出
			lw	M	地址未字对齐
			lh	M	地址未半字对齐
			LOAD	M	地址不在存取合法地址区段
			lh, lb	M	地址在 TC 地址区段
5	AdES	存数异常	STORE	E	计算地址时加法溢出
			sw	M	地址未字对齐
			sh	M	地址未半字对齐
			STORE	M	地址不在存取合法地址区段
			STORE	M	地址为 TC_Count 地址
			sh, sb	M	地址在 TC 地址区段
8	Syscall	系统调用	syscall	D	执行 syscall 指令
10	RI	未知指令	(Other)	D	执行未知指令
12	Ov	算术溢出	add addi sub	E	执行考虑溢出的算数指令时发生溢出

注：

1.流水级为判断是否发生该异常的流水级。

2.多种中断异常时，外部中断覆盖所有异常，流水级在前的异常覆盖流水级在后的异常。

3.发生取指异常时，受害指令是 PC 值不正确的指令，需要向 EPC 写入不正确的地址

4.发生取指异常时，视为 nop 直到提交到 CP0。

注意官方 Testbench 给出的取指相关语句是：

```
reg [31:0] inst[0:5119];
assign i_inst_rdata = inst[((i_inst_addr - 32'h3000) >> 2) % 5120];
```

虽然在初始化时会把 inst 全部清零，但是在发生取值异常时，**取出的指令不一定是 nop，需要手动置为 nop**。

5.存取合法地址区段共 4 段：

0x0000-0x2fff、0x7f00-0x7f0b、0x7f10-0x7f1b、0x7f20-0x7f23

TC 地址区段共 2 段：

0x7f00-0x7f0b、0x7f10-0x7f1b

TC_Count 地址区段共 2 段：

0x7f08-0x7f0b、0x7f18-0x7f1b

6.对于未知指令的判断仅需考虑 opcode 和 R 型指令的 funct。测试程序保证当指令的 opcode 正确，且如为 R 型指令其 funct 也正确时，指令其他部分格式一定正确。保证未知指令的 opcode 和 funct 组合一定没有在 MARS 基本指令集中出现。

7.发生未知指令异常时，视为 nop 直到提交到 CP0。

CP0 寄存器

名称	编号	读写性	功能
SR（State Register）	12	R/W	配置异常的功能
Cause	13	R	记录异常发生的原因和情况
EPC	14	R/W	记录异常处理结束后需要返回的 PC

注：

1.CP0 寄存器的初始值均为 0，**未实现位和未实现寄存器始终保持 0**。

2.测试程序保证不会写入 Cause。

SR：

31:16	15:10	9:2	1	0
0	IM	0	EXL	IE
Always 0	Interrupt Mask	Always 0	Exception Level	Interrupt Enable

注：当进入中断或异常状态时，由**硬件**将 EXL 置位，强制进入核心态并屏蔽中断信号；当退出中断或异常状态时，由 **eret 指令（软件）**将 EXL 清零。

Cause：

31	30:16	15:10	9:7	6:2	1:0
BD	0	IP	0	ExcCode	0
Branch Delay	Always 0	Interrupt Pending	Always 0	Exception Code	Always 0

注：

1.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。

2.无论是否允许中断 IP 域每周期都要写入外部硬件中断信号（HWInt）的值，是否进入中断或异常状态由 CP0 根据 SR_IM、SR_EXL、SR_IE 判断。

EPC：

31:0
EPC
Exception Program Counter

注：当该 Cause_BD 为 1 时，即发生异常或中断时提交点的指令为分支跳转指令的延迟槽中的指令，EPC 指向前一条指令（PC-4，即分支跳转指令），否则指向当前指令。

CP0

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
WE	I	写入使能信号
A[4:0]	I	操作的协处理器寄存器的编号
D[31:0]	I	写入协处理器寄存器的数据
D_BD	I	当前 M 级指令是否为分支跳转指令的延迟槽指令
D_VPC[31:0]	I	当前 M 级指令的地址，可能为错误的地址
D_ExcCode[4:0]	I	当前 M 级指令产生的异常
D_HWInt[5:0]	I	当前外部中断产生的情况
EXL_CLR	I	下一周期清零 SR_EXL 信号
Req	O	进入中断处理程序信号
Q[31:0]	O	读取的协处理器寄存器的值

Q_EPC[31:0]	O	当前 EPC 寄存器的值
Q_EXL	O	当前 SR_EXL 的值

ER_D、ER_E、ER_M

通用：

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
Req	O	进入中断处理程序信号

以下端口均有 I 和 O 两个方向，I 方向信号名以 D_开头，O 方向信号名以 Q_开头。

ER_D：

端口名	接入
STALL_EN_N	CTRL_S_D_EN_N
ERET	CTRL_D_ERET
EPC[31:0]	CP0_Q_EPC
BD	CTRL_D_ISBRANCHJUMP
VPC[31:0]	IFU_InstrAddr
ExcCode[4:0]	ECMUX_F_ExcCode_F

ER_E：

端口名	接入
BD	ER_D_BD
VPC[31:0]	ER_D_VPC
ExcCode[4:0]	ECMUX_D_ExcCode_D

ER_M：

端口名	接入
BD	ER_E_BD
VPC[31:0]	ER_E_VPC
ExcCode[4:0]	ECMUX_E_ExcCode_E

注：

1.各级 ER 中 **VPC 的复位值为 32'h0000_3000**，维护宏观 PC，确保在暂停过程中出现中断时 EPC 正常写入数据。

2.各级 ER 中 ExcCode 为 5'd0 时表示在该级之前没有发生异常。

ECMUX_F、ECMUX_D、ECMUX_E、ECMUX_M、

ECMUX_F：

端口名	方向	描述
InstrAddr[31:0]	I	即将进入流水线的指令地址
ExcCode_F[4:0]	O	F 级产生的异常的 ExcCode
AdEL_F	O	F 级是否产生取值异常

ECMUX_D：

端口名	方向	描述
ExCode_F[4:0]	I	F 级产生的异常的 ExcCode
SYSCALL	I	D 级指令是否为 syscall

UNKNOWN	I	D 级指令是否为未知指令
ExcCode_D[4:0]	O	D 级产生的异常的 ExcCode

ECMUX_E：

端口名	方向	描述
ExCode_D[4:0]	I	D 级产生的异常的 ExcCode
Overflow	I	E 级指令是否发生算数溢出异常
ISLOADSTORE	I	E 级指令是否为系统桥操作类指令
BridgeSel[3:0]	I	E 级系统桥操作指令操作模式
ExcCode_E[4:0]	O	E 级产生的异常的 ExcCode

ECMUX_M：

端口名	方向	描述
ExCode_E[4:0]	I	E 级产生的异常的 ExcCode
AdE[1:0]	I	M 级指令是否发生取数异常(2'b10)或存数异常(2'b11)
ExcCode_M[4:0]	O	M 级产生的异常的 ExcCode

宏观 PC

宏观 PC 为 M 级正在处理的指令。从 ER_M_VPC 引出。

进入中断处理程序操作

1.D_ExcCode 不为 0 或 D_HWInt 和 SR_IM 与运算后不为 0，且 SR_EXL 为 0，SR_IE 为 1，产生 Req 信号。

2.1 根据多种中断异常时，外部中断覆盖所有异常，流水级在前的异常覆盖流水级在后的异常的原则确定 ExcCode，下一周期写入 Cause_ExcCode。

2.2 根据 D_BD 和 D_VPC 确定 EPC，下一周期写入 Cause_BD 和 EPC。

2.3 让 SR_EXL 下一周期置位。

2.4 禁止对于协处理器寄存器的写入。

3.让 NPC 输出 0x4180，使 PC 下一周期跳转到中断处理程序开头位置。

4.让 FR_D、FR_E、FR_M、FR_W 下一周期清空数据，即清空流水线。

5.让 ER_D、ER_E、ER_M 的 VPC 下一周期变为 0x4180，清空 BD 和 ExcCode，维护宏观 PC。

6.禁止 MULT 启动乘除法计算和 HI/LO 的写入，禁止 BRIDGE 的写入操作执行。

离开中断处理程序操作

1.执行到中断处理程序的 eret，产生 ERET 信号。

2.让 NPC 输出 EPC，使 PC 下一周期跳转到 EPC 的位置。

3.让 FR_D 下一周期清空数据，等价于将指令变成 nop，实现 eret 指令没有延迟槽。

4.让 ER_D 的 VPC 下一周期变为 EPC，清空 BD 和 ExcCode，维护宏观 PC，确保在暂停过程中出现中断时 EPC 正常写入数据。

5.流水 eret 信号到 M 级，通过 EXL_CLR 将 SR_EXL 位清零。

注：

1.允许测试程序从 0x417c 直接前进到 0x4180，即在正常状态下执行异常处理程序，此种情况下不应有中断响应等其他行为，故是否产生 eret 信号要结合 SR 寄存器中的 EXL 位判断。

2.测试程序保证包含异常的情况不会由 0x417c 直接前进到 0x4180 且 eret 只会出现在中断处理程序中，即不会出现以下情况：

```
.ktext
……
eret
eret
```

正常状态执行到第一个 eret 时产生了中断，执行中断处理程序到第一个 eret 返回。执行序列为第一个 eret（异常状态）->第一个 eret（正常状态）->第二个 eret（正常状态）。当第二个 eret（正常状态）进入到 D 级时，第一个 eret（正常状态）才刚进入到 M 级，还未完成对 EXL 的修改，这样第二个 eret（正常状态）还是会引发跳转，最终导致死循环。

CPU 层功能模块设计

暴露至 MIPS 体系结构层的端口

端口名	方向	外部信号（I 方向）	内部信号（O 方向）
CPU_RESET	I	reset	
CPU_clk	I	clk	
CPU_macroscopic_pc[31:0]	O		ER_M_VPC
CPU_IFU_InstrAddr[31:0]	O		IFU_InstrAddr
CPU_IFU_Instr[31:0]	I	i_inst_rdata	
CPU_GRFWE_W	O		GRFWE_W
CPU_GRF_A3_W[4:0]	O		GRF_A3_W
CPU_GRF_WD_W[1:0]	O		MUX_GRF_WD_W
CPU_Exam_InstrAddr_W[31:0]	O		Exam_InstrAddr_W
CPU_Req	O		CP0_Req
CPU_BridgeSel_M[3:0]	O		BridgeSel_M
CPU_ISLOADSTORE_M	O		ISLOADSTORE_M
CPU_BRIDGE_A[31:0]	O		E_RES_M
CPU_BRIDGE_D[31:0]	O		FMUX_V2_M
CPU_BRIDGE_AdE[1:0]	I	BRIDGE_AdE	
CPU_BRIDGE_HWInt[5:0]	I	BRIDGE_HWInt	
CPU_BRIDGE_Q[31:0]	I	BRIDGE_Q	

IFU

1. 描述

取指令模块。内有 PC 子模块、NPC 子模块。接收分支跳转指令控制信号、中断处理程序进入退出控制信号和相关数据，输出指令地址，通过与外部 ROM 通信获取指令。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
STALL_EN_N	I	暂停使能信号
Req	I	进入中断处理程序信号
ERET	I	退出中断处理程序信号
NPCSel[1:0]	I	NPC 子模块控制信号
BranchComp	I	NPC 子模块分支指令控制信号
offset[15:0]	I	分支指令跳转偏移
instr_index[25:0]	I	j、jal 指令跳转地址
instr_register[31:0]	I	jr、jalr 指令跳转寄存器
EPC[31:0]	I	退出中断处理程序返回地址
InstrAddr[31:0]	O	即将进入流水线的指令地址

3. 控制信号

BranchComp：接 COMP 的 BranchComp，表示分支指令的条件是否满足。

NPCSel：

指令（信号）名	取值	描述
其他	00	PC←PC+4
BRANCHE、BRANCHZ	01	使用 offset，结合 BranchComp 判断
j、jal	10	使用 instr_index
jr、jalr	11	使用 instr_register

注：Req 和 ERET 对 NPC 的影响的优先级高于 NPCSel。

4. 关联 MUX

EMUX_Instr，控制信号 ECMUX_F_AdEL_F

控制信号取值	数据	描述
0	IFU_Instr	没有产生取指异常，正常输出指令
1	32'h00000000 (nop)	产生取指异常，强制改成 nop

IFU_PC

1. 描述

程序计数器子模块。在复位信号有效时复位为 32'h00003000。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
STALL_EN_N	I	暂停使能信号
D[31:0]	O	下一指令地址
Q[31:0]	O	即将进入流水线的指令地址

IFU_NPC

1. 描述

下一条指令地址计算子模块。输入 IFU 接收分支跳转指令控制信号、中断处理程序进入退出控制信号和相关数据，计算出下一条指令的地址。

2. 接口

端口名	方向	描述
Req	I	进入中断处理程序信号
ERET	I	退出中断处理程序信号
PC[31:0]	1	即将进入流水线的指令地址
NPCSel[1:0]	I	同 IFU
BranchComp	I	同 IFU
offset[15:0]	I	同 IFU
instr_index[25:0]	I	同 IFU
instr_register[31:0]	I	同 IFU
EPC[31:0]	I	同 IFU
NPC[31:0]	O	下一指令地址

SPL

1. 描述

指令分线器模块。按照三种指令的格式将一条指令分为不同的部分，供其他模块使用。

若 M 级发生未知指令异常，将指令强制改成 nop。

2. 接口

端口名	方向	描述
force_nop	I	发生未知指令异常，强制改成 nop
Instr[31:0]	1	当前指令
O_opcode[5:0]	O	未经强制 nop 的指令的操作数
O_funct[5:0]	O	未经强制 nop 的 R 型指令的 funct
O_rs[4:0]	O	未经强制 nop 的 R、I 型指令的 rs
O_rt[4:0]	O	未经强制 nop 的 R、I 型指令的 rt
rs[4:0]	O	经强制 nop 的 R、I 型指令的 rs
rt[4:0]	O	经强制 nop 的 R、I 型指令的 rt
rd[4:0]	O	经强制 nop 的 R 型指令的 rd
shamt[4:0]	O	经强制 nop 的 R 型指令的 shamt
imm16[15:0]	O	经强制 nop 的 I 型指令的立即数
instr_index[25:0]	O	经强制 nop 的 J 型指令的跳转地址

GRF
(GRF_R、GRF_W)

1. 描述

寄存器堆模块，共有 31 个寄存器（0 号寄存器直接接地）。通过 A1、A2 输入地址，可以取出指定寄存器的完成内部转发的数据。当 WE 使能时，通过 A3 输入地址，D 输入数据，可以修改指定寄存器存储的数据并进行内部转发。在流水线设计时分为两个部分体现。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
WE	I	写入使能信号
A1[4:0]	I	读取的第一个寄存器的编号
A2[4:0]	I	读取的第二个寄存器的编号
A3[4:0]	I	写入的寄存器的编号
WD[31:0]	I	写入寄存器的数据
V1[31:0]	O	读取的第一个寄存器的完成内部转发的值
V2[31:0]	O	读取的第二个寄存器的完成内部转发的值

3. 控制信号

GRFWE：

指令信号名	取值	描述
其他	0	不涉及寄存器写入的指令
ALREG、ALIMM、EXTLUI	1	
MLFROM	1	
LOAD	1	
JUMPW、JUMPRW	1	
MFCOP	1	

4. 关联 MUX

MUX_GRF_A3_D，控制信号 CTRL_D_GRF_A3_D_Sel

指令信号名	取值	描述
其他	00	接入 rd
ALIMM、EXTLUI	10	接入 rt
LOAD、MFCOP	10	接入 rt
JUMPW	11	固定 32’d31

MUX_GRF_WD_W，控制信号 GRF_WD_W_Sel_W

指令信号名	取值	描述
其他	00	接入 E_RES_W
LOAD、MFCOP	01	接入 M_RES_W
EXTLUI	10	接入 ext32_W
JUMPW、JUMPRW	11	接入 pc8_W

COMP

1. 描述

比较模块，完两数相等、不等，一数有符号小于、小于等于、大于、大于等于零。比较结果用于判断分支指令是否执行。

2. 接口

端口名	方向	描述
CompSel[2:0]	I	比较方式控制信号
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
BranchComp	O	比较结果

3. 控制信号

CompSel：

指令名	取值	描述
bltz(00000)	000	有符号小于 0
bgez(00001)	001	有符号大于等于 0
beq(000100)	100	两数相等
bne(000101)	101	两数不等
blez(000110)	110	有符号小于等于 0
bgtz(000111)	111	有符号大于 0

EXT

1. 描述

位扩展模块，将 16 位立即数无符号扩展、有符号扩展、右补 16 位 0 为 32 位。

2. 接口

端口名	方向	描述
EXTSel[1:0]	I	扩展方式控制信号
imm16[15:0]	I	16 位立即数
ext32[31:0]	O	32 位扩展结果

3. 控制信号

EXTSel：

指令名	取值	描述
其他	00	有符号拓展
andi、ori、xori	01	无符号拓展
lui	10	右补 16 位 0

ALU

1. 描述

算数逻辑模块，完成算数运算（不考虑算数溢出、考虑算数溢出的加、减）、逻辑运算（与、或、异或、或非）、移位运算（左移、逻辑右移、算数右移）、比较运算（两数有符号小于，无符号小于）。计算结果用于保存到寄存器。

2. 接口

端口名	方向	描述
OPSel[1:0]	I	OP 输出控制信号
FuncSel[1:0]	I	计算方式控制信号
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
shamt[4:0]	I	移位立即数
OP[31:0]	O	计算结果
Overflow	O	计算结果是否出现算数溢出

3. 控制信号

OPSel：

指令（信号）名	取值	描述
add、addu、sub、subu、addi、addiu	00	算数运算
LOAD、STORE	00	算数运算
and、or、xor、nor、andi、ori、xori	01	逻辑运算
sll、srl、sra、sllv、srlv、srav	10	移位运算
slt、sltu、slti、sltiu	11	比较运算

FuncSel：接 CTRL，根据 OPSel 选择。

OPSel == 00：

指令名	取值	描述
addu、addiu	00	不考虑算数溢出加法
subu	01	不考虑算数溢出减法
add、addi	10	考虑算数溢出加法
LOAD、STORE	10	考虑算数溢出加法
sub	11	考虑算数溢出减法

注：默认值应为 00，避免意外触发算数溢出异常。

OPSel == 01：ALREG 接 funct[1:0]、ALIMM 时接 opcode[1:0]

指令名	取值	描述
and(100100)、andi(001100)	00	与
or(100101)、ori(001101)	01	或
xor(100110)、xori(001110)	10	异或
nor(100111)	11	或非

OPSel == 10：接 funct[1:0]。

指令名	取值	描述
sll(000000)、sllv(000100)	00	逻辑左移
srl(000010)、srlv(000110)	10	逻辑右移
sra(000011)、srav(000111)	11	算数右移

OPSel == 11：（只用设置 01）

指令名	取值	描述
slt、slti	00	有符号两数小于
sltu、sltiu	01	无符号两数小于

4. 关联 MUX

MUX_ALU_B_E，控制信号 ALU_B_E_Sel_E

指令信号名	取值	描述
其他	0	接入 GRF
ALIMM、LOAD、STORE	1	接入 EXT

MULT

1. 描述

乘除法模块，执行乘法的时间为 5 个时钟周期，执行除法的时间为 10 个时钟周期（包含写入内部的 HI、LO 寄存器）。自 1 个时钟周期 Start 信号有效后的第 1 个 clock 上升沿开始乘除法运算，同时 Busy 置位。在运算结果保存到 HI 寄存器和 LO 寄存器后，Busy 清零。写入 HI、LO 均只需 1 个时钟周期，类似 GRF。读取 HI、LO 直接读取，类似 ALU。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
Req	I	进入中断处理程序信号
ISMULTDIV	I	是否为乘除指令信号
MULTSel[1:0]	I	乘除模块控制信号
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
Start	O	乘除计算启动信号，输出 ISMULTDIV & MULTSel[2]
Busy	O	乘除计算进行信号
HILO[31:0]	O	读取的 HI、LO 的结果

3. 控制信号

ISMULTDIV：指令信号名为 MULTDIV、MLTO、MLFROM 时为 1.

MULTSel：接{funct[3], funct[1:0]}。

指令名	取值	描述
mult(011000)	100	准备执行**有符号**乘法操作
multu(011001)	101	准备执行**无符号**乘法操作
div(011010)	110	准备执行**有符号**除法操作
divu(011011)	111	准备执行**无符号**除法操作
mfhi(010000)	000	HILO 输出 HI
mflo(010010)	010	HILO 输出 LO
mthi(010001)	001	准备将 A 写入 HI

mtlo(010011)	011	准备将 A 写入 LO

4. 关联 MUX

MUX_E_RES_E，控制信号 E_RES_E_Sel_E：

指令信号名	取值	描述
其他	0	接入 ALU_OP
MLFROM	1	接入 MULT_HILO

5. 中断异常行为

1.MULTDIV 在 E 级启动了乘法运算，流水到 M 级时产生了中断，此时无需停止乘法计算。

2.MLTO 在 E 级修改了 HI/LO，流水到 M 级时产生了中断，此时无需恢复 HI/LO 的值。

3.MULTDIV 在 E 级（还未改变 MDU 状态），受害指令在 M 级，此时不应开始乘法计算。

4.MLTO 在 E 级（还未改变 MDU 状态），受害指令在 M 级，此时不应修改 HI/LO 的值。

（BRIDGE）

1. 关联 MUX

MUX_M_RES_M，控制信号 ISLOADSTORE_M：

指令信号名	取值	描述
其他	0	接入 CP0_Q
LOAD、STORE	1	接入 CPU_BRIDGE_Q

FR_D、FR_E、FR_M、FR_W

1. 描述

D、E、M、W 段流水线寄存器。

2. 接口

通用：

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
Req	I	进入中断处理程序信号

以下端口均有 I 和 O 两个方向，I 方向信号名以 D_开头，O 方向信号名以 Q_开头。

FR_D：

端口名	接入
STALL_EN_N	CTRL_S_D_EN_N
ERET	CTRL_D_ERET
Instr[31:0]	EMUX_Instr
InstrAddr[31:0]	IFU_InstrAddr

注：FR_D 的 ERET 是控制信号。

FR_E：

端口名	接入
STALL_RESET	CTRL_S_FR_E_RESET
ISLOADSTORE	CTRL_D_ISLOADSTORE
GRFWE	CTRL_D_GRFWE
BridgeSel[3:0]	CTRL_D_BridgeSel
FuncSel[1:0]	CTRL_D_FuncSel

OPSel[1:0]	CTRL_D_OPSel
GRF_WD_W_Sel[1:0]	CTRL_D_GRF_WD_W_Sel
ALU_B_E_Sel	CTRL_D_ALU_B_E_Sel
E_RES_E_Sel	CTRL_D_E_RES_E_Sel
MULTSel[2:0]	CTRL_D_MULTSel
ISMULTDIV	CTRL_D_ISMUTLDIV
CP0WE	CTRL_D_CP0WE
ERET	CTRL_D_ERET
V1[31:0]	FMUX_V1_D
V2[31:0]	FMUX_V2_D
rd[4:0]	SPL_rd
shamt[4:0]	SPL_shamt
GRF_A3[4:0]	MUX_GRF_A3_D
ext32[31:0]	EXT_ext32
pc8[31:0]	pc8_D
FMUX_V1_E_Sel[1:0]	CTRL_F_FMUX_V1_E_Sel
FMUX_V2_E_Sel[1:0]	CTRL_F_FMUX_V2_E_Sel
FMUX_V2_M_Sel	CTRL_F_FMUX_V2_M_Sel

注：FR_E 的 ERET 是流水数据。

FR_M：

端口名	接入
ISLOADSTORE	ISLOADSTORE_E
GRFWE	GRFWE_E
BridgeSel[3:0]	BridgeSel_E
GRF_WD_W_Sel[1:0]	GRF_WD_W_Sel_E
CP0WE	CP0WE_E
ERET	ERET_E
V2[31:0]	FMUX_V2_E
rd[4:0]	rd_E
E_RES[31:0]	MUX_E_RES_E
GRF_A3[4:0]	GRF_A3_E
ext32[31:0]	ext32_E
pc8[31:0]	pc8_E
FMUX_V2_M_Sel	FMUX_V2_M_Sel_E

FR_W：

端口名	接入
GRFWE	GRFWE_M
GRF_WD_W_Sel[1:0]	GRF_WD_W_Sel_M
E_RES[31:0]	E_RES_M
M_RES[31:0]	MUX_M_RES_M
GRF_A3[4:0]	GRF_A3_M
ext32[31:0]	ext32_M
pc8[31:0]	pc8_M

Exam_InstrAddr[31:0]	ER_M_VPC

CTRL_Decoder

1. 描述

   译码器模块。通过 opcode、funct、rs 和 rt 判断指令的类型，输出各个模块的控制信号。

2. 接口

端口名	方向	端口名	方向
opcode[5:0]	I	MULTSel[2:0]	O
funct[5:0]	I	ISMULTDIV	O
rs[4:0]	I	ISLOADSTORE	O
rt[4:0]	I	BridgeSel[3:0]	O
CP0_Q_EXL	I	CP0WE	O
SYSCALL	O	GRFWE	O
UNKNOWN	O	GRF_A3_D_Sel[1:0]	O
ISBRANCHJUMP	O	ALU_B_E_Sel	O
ERET	O	E_RES_E_Sel	O
CompSel[2:0]	O	GRF_WD_W_Sel[1:0]	O
EXTSel[1:0]	O	Tuse_rs[1:0]	O
NPCSel[1:0]	O	Tuse_rt[1:0]	O
OpSel[1:0]	O		
FuncSel[1:0]	O		

CTRL_Stall

1. 描述

   暂停控制器模块。判断是否需要暂停，若需要暂停则产生暂停信号。

2. 接口

端口名	方向	端口名	方向
Tuse_rs[1:0]	I	GRF_A3_E[4:0]	I
Tuse_rt[1:0]	I	GRF_A3_M[4:0]	I
SPL_rs[4:0]	I	ISMULTDIV	I
SPL_rt[4:0]	I	MULT_Start	I
GRFWE_E	I	MULT_Busy	I
GRFWE_M	I	IFU_EN_N	O
GRF_WD_W_Sel_E[1:0]	I	D_EN_N	O
GRF_WD_W_Sel_M[1:0]	I	FR_E_RESET	O

CTRL_Forward

1. 描述

   转发控制器模块。判断是否需要转发，若需要转发转发什么数据。

2. 接口

端口名	方向	端口名	方向
Tuse_rs[1:0]	I	GRF_A3_E[4:0]	I
Tuse_rt[1:0]	I	GRF_A3_M[4:0]	I

SPL_rs[4:0]	I	FMUX_V1_D_Sel[2:0]	O
SPL_rt[4:0]	I	FMUX_V2_D_Sel[2:0]	O
GRFWE_E	I	FMUX_V1_E_Sel[1:0]	O
GRFWE_M	I	FMUX_V2_E_Sel[1:0]	O
GRF_WD_W_Sel_E[1:0]	I	FMUX_V2_M_Sel	O
GRF_WD_W_Sel_M[1:0]	I		

MIPS 体系结构层功能模块设计

暴露至外部设备层的端口

端口名	方向	内部信号
clk	I	CPU_clk、TC0_clk、TC1_clk
reset	I	CPU_RESET、TC0_RESET、TC1_RESET
interrupt	I	BRIDGE_HWInt_2
macroscopic_pc[31:0]	O	CPU_macroscopic_pc
i_inst_addr[31:0]	O	CPU_IFU_InstrAddr
i_inst_rdata[31:0]	I	CPU_IFU_Instr
m_data_addr[31:0]	O	CPU_BRIDGE_A
m_data_rdata[31:0]	I	BRIDGE_rdata_DM
m_data_wdata[31:0]	O	BRIDGE_wdata
m_data_byteen[3:0]	O	BRIDGE_byteen_DM
m_int_addr[31:0]	O	CPU_BRIDGE_A
m_int_byteen[3:0]	O	BRIDGE_byteen_2
m_inst_addr[31:0]	O	CPU_macroscopic_pc
w_grf_we	O	CPU_GRFWE_W
w_grf_addr[4:0]	O	CPU_GRF_A3_W
w_grf_wdata[31:0]	O	CPU_GRF_WD_W
w_inst_addr[31:0]	O	CPU_Exam_InstrAddr_W

BRIDGE

1. 描述

系统桥模块。连接 CPU 与 DM、Interrupt Generator、Timer0、Timer1 等外部设备，将 LOAD、STORE 指令根据地址图分发到各个外部设备，判断是否产生 AdE 异常，收集外部设备的中断请求并集中发送给 CPU。

2. 接口

端口名	方向	描述
ISLOADSTORE	I	是否为系统桥操作类指令
BridgeSel[3:0]	I	系统桥操作指令操作模式
Req	I	进入中断处理程序信号
A[31:0]	I	读取或写入的地址
D[31:0]	I	原始的写入数据
rdata_DM[31:0]	I	DM 原始的读取数据
rdata_(0-5)[31:0]	I	共 6 个，外部设备原始的读取数据
HWInt_(0-5)	I	共 6 个，外部设备的中断信号
wdata[31:0]	O	处理后的写入数据
byteen_DM[3:0]	O	DM 字节使能信号
byteen_(0-5)[3:0]	O	共 6 个，外部设备的字节使能信号
AdE[1:0]	O	是否产生 AdE 异常以及产生种类
HWInt_Hub[5:0]	O	收集的所有外部设备 HWInt 中断信号
Q[31:0]	O	处理后的读取数据

注：Req 高电平时，即将进入中断处理程序时，禁止写入。

3. 控制信号

ISLOADSTORE：指令信号名为 LOAD、STORE 时为 1。只有其为 1 时才可以执行判断异常和写入操作。

BridgeSel：接 opcode[3:0]。

指令名	取值	描述
lb(100000)	0000	以 byte 读取，结果有符号拓展
lh(100001)	0001	以 halfword 读取，结果有符号拓展
lw(100011)	0011	以 word 读取
lbu(100100)	0100	以 byte 读取，结果无符号拓展
lhu(100101)	0101	以 halfword 读取，结果无符号拓展
sb(101000)	1000	以 byte 写入
sh(101001)	1001	以 halfword 写入
sw(101011)	1011	以 word 写入

4. 输出信号

byteen_DM、byteen_(0-5)：

指令名	BridgeSel	A[1:0]	byteen	A[1:0]	byteen	A[1:0]	byteen	A[1:0]	byteen
其他		00	0000	01	0000	10	0000	11	0000
sb	1000		0001		0010		0100		1000
sh	1001		0011		/		1100		/
sw	1011		1111		/		/		/

注：字节使能信号表示 D 的哪些字节要被写入，若要将低位的数据写入高位**需要进行移位**。

AdE：

异常类型	AdE
无异常	00
AdEL	10
AdES	11

5. 地址图

设备	地址范围	说明
数据存储器（DM）	0x0000_0000~0x0000_2FFF	
指令存储器（IM）	0x0000_3000~0x0000_6FFF	
计时器 0 寄存器（Timer0）	0x0000_7F00~0x0000_7F0B	计时器 0 共有 3 个寄存器
计时器 1 寄存器（Timer1）	0x0000_7F10~0x0000_7F1B	计时器 1 共有 3 个寄存器
中断发生器响应地址	0x0000_7F20~0x0000_7F23	

注：测试程序只会通过指令 `sb $0, 0x7f20($0)`访问中断发生器以模拟响应中断，且只会在中断处理程序中访问。

6. 外部设备连接

信号路	外部设备	rdata	byteen	HWInt
DM	DM	m_data_rdata	m_data_byteen	/
0	Timer0	TC0_rdata	TC0_byteen	TC0_IRQ
1	Timer1	TC1_rdata	TC1_byteen	TC1_IRQ
2	Interrupt Generator	接地	m_int_byteen	interrupt
3	无	接地	输出 0，浮空	接地

4	无	接地	输出 0，浮空	接地
5	无	接地	输出 0，浮空	接地

注：Interrupt Generator 并没有真正的存储单元，规定读出的数据始终保持 0。

Timer0、Timer1

1. 描述

计时器模块。由课程组设计，有两种模式。模式 0 用于产生定时中断，模式 1 用于产生周期性脉冲，具体见课程组提供的 PDF 文件《COCO 定时器设计规范》。

注：TC 的寄存器的值只允许以字的形式读取写入，否则将引发 AdE 错误。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
byteen[3:0]	I	字节使能信号
addr[31:0]	I	操作的计时器寄存器的地址
wdata[31:0]	I	写入计时器寄存器的数据
rdata[31:0]	O	读取的计时器寄存器的值
IRQ	O	中断请求信号

P7 思考题

1. **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？**

通过外部中断的机制。鼠标或键盘内部有一些寄存器，保存鼠标或键盘最近一段时间的操作信息。当产生了新的操作时，产生外部中断，CPU 通过访问特定的地址可以访问到鼠标或键盘的寄存器，经过分析可以得到鼠标或键盘的操作。

2. **请思考为什么我们的 CPU 处理中断异常必须是已经指定好的地址？如果你的 CPU 支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）**

理论上可以实现希望的功能。但是若实现不知道中断处理程序的地址，可能在正常执行程序时会对存放中断处理程序的区域进行误操作读写，从而使中断处理程序被破坏。且支持自定义中断处理程序的入口地址也会增加硬件设计难度。

3. **为何与外设通信需要 Bridge？**

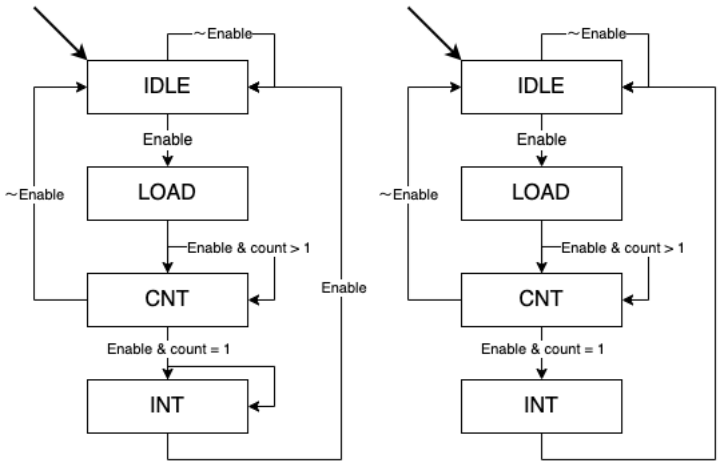
使 CPU 以统一的方式访问不同种类的外部设备。在增加新的外部设备时也更加方便，不需要影响 CPU 的既有结构，符合“高内聚低耦合”的原则。

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移图。**

模式 0：当计数器倒计数为 0 后，计数器停止计数，持续产生中断信号，当使能被再次置位后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数。通常用于产生定时中断。

模式 1：当计数器倒计数为 0 后，只产生一周期中断信号，初值寄存器值被自动加载至计数器，计数器继续倒计数。通常用于产生周期性脉冲。

模式 0 状态转移图如下图左图所示，模式 0 状态转移图如下图右图所示。



5. **倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的 CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息？**

丢失是否为延迟槽指令和受害 PC 信息，将导致 EPC 写入错误的数值。实际上，在进入异常中断程序而清空流水线的情况下，无需保留任何信息，但是要注意将受害 PC 置位中断处理程序的开始的地址（在 P7 中即 0x4180）。若是因为流水线暂停而导致在 E 级插入的 nop，应该要注意让 E 级流水 D 级是否为延迟槽指令和受害 PC 信息。

6. **为什么 jalr 指令为什么不能写成 jalr $31, $31？**

若 jalr 读写同一寄存器，则会将 PC + 8 写入该寄存器。离开中断处理程序后再次执行这条指令的效果与之前不一致。

在 MIPS 指令集中也有相关约束：Register specifiers rs and rd must not be equal, because such an instruction does not have the same effect when reexecuted. The result of executing such an instruction is **UNPREDICTABLE**. This restriction permits an exception handler to resume execution by re-executing the branch when an exception occurs in the branch delay slot.