CPU 设计文档

目录：

一、相关规定与设计要求

1.设计规格说明

（1）MIPS 微系统包括 MIPS 处理器（CPU）、系统桥（bridge）和 2 个定时器（timer），支持中断和异常，需要实现系统桥（bridge）和 2 个设备（device），device 为定时器（timer）。

（2）CPU 支持 MIPS-lite2-int 指令集：{ADDU、SUBU、**ADD**、**SUB**、ORI、LW、SW、BEQ、LUI、J、JAL、JR、NOP、**ERET**、**MFC0**、**MTC0**}，为流水线设计，**eret 指令需要由硬件实现后跟 nop，即当 eret 指令到达 E 级时清除 D 级指令**。

（3）本 project 需要支持的异常：

ExcCode	助记符	描述
0	Int	中断
4	AdEL	取数或取指时地址错误
5	AdES	存数时地址错误
10	RI	不认识的（或者非法的）指令码
12	Ov	自陷形式的整数算术指令（例如add）导致的溢出

（4）为了支持异常和中断，流水线必须支持 0 号协处理器（CP0），必须实现的 CP0 寄存器包括:**SR**、**CAUSE**、**EPC**、**PrID**。

（5）MIPS 的异常入口只支持 **0x00004180**，所有的异常与中断都从这里进入，即 **2 个 timer 的异常入口是完全相同的**。

2.设计约束

（1）IM 和 DM：容量为 16KB（32bit/word×4Kword）。

（2）整个设计的顶层模块应该至少包括：CPU、bridge 和 2 个 timer。

CPU：在 P5 基础上添加相应扩展。

bridge：作为独立 module，不包括在 CPU 中。

（3）地址空间：

	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

（4）Exception handler 的代码属于指令存储器。

（5）timer0 输出的中断请求信号请接入 MIPS 处理器的 HWInt[2]，即最低中断位。timer1 输出的中断请求信号请接入 MIPS 处理器的 HWInt[3] 。

3.中断异常行为规范

（1）对于异常的处理，遵循精确异常的处理规则，受害指令的前序指令的结果保留，受害指令的后序指令的效果不影响异常返回后后序指令执行时的效果。

（2）对于中断的发生，不指定哪条为受害指令，只需保证执行中断处理程序前后每条指令的执行效果不变即可。

（3）当异常或中断发生时，应立即将 EXL 置为 1，表示屏蔽所有中断，并将 ExcCode 写入到 Cause 寄存器中。

（4）中断的优先级高于异常的优先级，不支持中断异常嵌套的情况。在中断异常处理程序结束后（即 eret 指令后），即可支持中断异常。

（5） 当 jal 的延迟槽指令发生异常时，jal 指令写入结果保存。当异常处理结束后，jal 应再次写入。

二、模块及数据通路增改

1.异常检测模块

异常检测模块用于检测除 W 外每一级的异常，产生 ExcCode 并存入本级流水寄存器。其中 F 级同时产生标记延迟槽指令的 BD 信号，用于 CP0 处理延迟槽指令异常。

表 2.1.1 各级异常类型及描述

级次	异常类型	异常描述
F	取指异常 AdEL	PC 值没有以 4 对齐 范围超出 0x3000~0x4ff。
	非法指令码 RI	IFU 取出一条非给定指令，即 op 及 fun 是否为给定
E	运算溢出 Ov	add、sub 引起的溢出
	取数异常 AdEL	lw 计算地址时加法溢出。
	存数异常 AdES	sw 计算地址时加法溢出。
M	取数异常 AdEL	lw 取数指令时，地址没有按照要求对齐 范围超出 0x0000~0x2ffc、0x7F00~0X7F0B、 0x7F10~0X7F1B
	存数异常 AdES	sw 取数指令时，地址没有按照要求对齐 范围超出 0x0000~0x2ffc、0x7F00~0X7F0B、 0x7F10~0X7F1B 试图往定时器的 COUNT 寄存器写入值

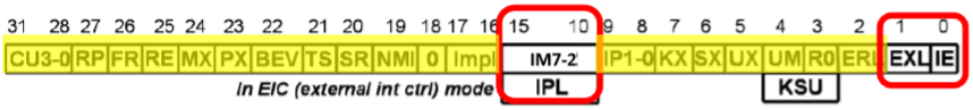
M 级由 bridge 传入 timer 的中断信号 HWInt，与 ExcCode 一起进入 CP0 被处理。

2.CP0 模块

CP0 主要用于处理中断异常信息，需求信息都来自 M 级流水寄存器，因此加入 M 级。支持 SR、Cause、EPC、PRId 四个寄存器，模块规格参考课件上的设计，并加入了 BD 信号。

图 2.2.1 12 号 SR 寄存器与 13 号 Cause 寄存器

- IM[7:2]：6位中断屏蔽位，分别对应6个外部中断 ◆ 1-允许中断，0-禁止中断
- IE：全局中断使能 ◆ 1-允许中断；0-禁止中断
- EXL：异常级 ◆ 1-进入异常，不允许再中断；0-允许中断



- IP[7:2]：6位待决的中断位，分别对应6个外部中断
  - 记录当前哪些硬件中断正在有效
- ExcCode[6:2]：异常编码，记录当前发生的是什么异常

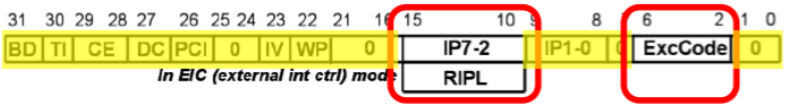


表 2.2.1 CP0 模块端口定义

端口名	方向	端口含义
[4:0] A1	I	读 CP0 寄存器的编号
[4:0] A2	I	写 CP0 寄存器的编号
[31:0] DIn	I	MTC0 指令写入 CP0 寄存器的值
[31:2] PC8	I	当前 M 级指令的 PC8 值
[6:2] ExcCode	I	M 级存储当前指令存在的异常类型
[5:0] HWInt	I	bridge 传入的中断信号
BD	I	M 级存储的当前指令的延迟槽指令标记
We	I	MTC0 的写使能信号
clk	I	时钟信号
rst	I	复位信号
EXLClr	I	EXL 清空信号，用于 eret 指令
IntReq	O	CP0 产生的中断异常处理信号
[31:0] EPC	O	向 NPC 传递 eret 所需的 EPC 地址
[31:0] DOut	O	MFC0 指令输出 CP0 寄存器的值

表 2.2. CP0 模块功能描述

序号	功能	功能描述
1	判断中断异常	IntReq = (|(HWInt & IM) & IE & !EXL)|

		((ExcCode != 5'b0) && !EXL)
2	处理中断	SR 存入 IM，EXL 与 IE，EXL 置位为 1 EPC 根据 BD 存入 PC-4 或 PC Cause 存入 HWInt，ExcCode 为 0（中断优先异常）
3	处理异常	SR 存入 IM，EXL 与 IE，EXL 置位为 1 EPC 根据 BD 存入 PC-4 或 PC Cause 存入 HWInt 与 ExcCode
4	输出寄存器值	根据 A1 的值输出对应寄存器值
5	存入寄存器值	写使能有效，根据 A2 的值向对应寄存器中存入值

3.bridge 模块

用于连接 CPU 与两个 timer，传递数据。

表 2.3.1 bridge 模块端口定义

端口名	方向	端口含义
[31:0] PrAddr	I	CPU 读设备的地址
[31:0] PrRD	O	设备传给 CPU 的数据
[31:0] PrWD	I	CPU 写给设备的数据
[3:2] DEV_Addr	O	设备可用的地址
[31:0] DEV_WD	O	CPU 写给设备的数据
[31:0] DEV1_RD	I	设备 1 向 CPU 输出的数据
[31:0] DEV2_RD	I	设备 2 向 CPU 输出的数据
Int0	I	设备 0 的中断信号
Int1	I	设备 1 的中断信号
[7:2] HWInt	O	传给 CPU 与 CP0 的待决中断位

4.timer 计时器

根据设定的时间来定时产生中断信号，用来模拟中断源。用图例状态机描述计时器行为，CPU 写入数据的优先级高于自改变的优先级。

表 2.4.1 timer 模块端口定义

端口名	方向	端口含义
clk	I	时钟信号

reset	I	复位信号
[3:2] Addr	I	设备内部偏移地址
We	I	写使能信号
[31:0] DataIn,	I	CPU 写入数据
[31:0] DataOut,	O	计数器向 CPU 输出的数据
IRQ	O	计时器发出的中断信号

5.datapath 修改

（1）IFU 修改

根据 CP0 产生的 IntReq，决定 IFU 的跳转是 handle 还是正常指令地址。

更改 IM 的大小与 PC 取指的地址宽度，增加 handler 在指令存储器中的地址位置，增加初始化。

如果发生取指异常或者 RI，则将指令改为 nop。

（2）NPC 修改

当跳转指令为 eret 时，跳转地址为 EPC 寄存器存储的值，且优先级高于 beq、jr、j 和 jal。

（3）DM 修改

更改 DM 的大小与地址的宽度。

（4）流水寄存器修改

D 级增加 BD 与 ExcCode，增加 eret 后加 nop 时清空 D 级的机制（PC 置位为 EPC），增加 IntReq 导致的清空机制。

E 级增加 BD 与 ExcCode，更改暂停时清空 E 级的机制（PC 置位为 D 级 PC 值），增加 IntReq 导致的清空机制。

M 级增加 BD 与 ExcCode，增加 IntReq 导致的清空机制。

W 级增加 IntReq 导致的清空机制。

（5）CP0 与 bridge 数据处理机制

做进行类似 DM 的处理，用指令信息（MTC0 与 MFC0）与 bridge 控制信号区分向 DM 读/写信息与向 CP0 或设备读/写信息。

（6）加入 bridge 带来的信息

CPU 与 bridge 的连接与 DM 相似，向 bridge 传入 ALU 计算出的地址与寄存

器 RD2 的值，得到设备传来的输出信息和中断信号。

添加 HITDM 信号，区分是向设备读写还是向 DM 读写。

6.mips 模块修改

将 bridge 与连个 timer 与 CPU 连接，每个 timer 的写使能信号为判断 bridge 得到的地址基地址是否为自身加上原本的 DM 写使能信号。

三、控制器设计

1.主控制器

在主控制器中加入新加指令的信号。

表 3.1.1 非转发暂停控制信号

控制信号	功能名称
branch	branch = 0 时，PC 端输入为 IFU.PC4； branch = 1 时，PC 端输入为 NPC。
Br	Br = 00 时，该指令为分支指令 beq，NPC = PC + 4 + sign_ext(Imm‖00)。 Br = 01 时，该指令为跳转指令 jal 或 j，NPC = {PC[31:28] ，Jal，0，0}。 Br = 10 时，该指令为跳转指令 jr，NPC = rs。 Br = 11 时，该指令为跳转指令 eret，NPC=EPC，优先级最高。
RegIn	RegIn = 00 时，数据写入 rt 寄存器； RegIn = 01 时，数据写入 rd 寄存器。 RegIn = 10 时，数据写入 31 号寄存器。
RegWr	RegWr = 0 时，GRF 不能写入数据； RegWr = 1 时，GRF 可以写入数据。
EXTOp	EXTOp = 00 时，进行高位 0 扩展，B = {16{0}，A}； EXTOp = 01 时，进行低位 0 扩展，B = {A，16{0}}； EXTOp = 10 时，进行符号扩展，B = {16{A[15]}，A}；。
ALUSrc	ALUSrc = 0 时，ALU 的第二个运算数来自转发复选器 MF_RD2_E； ALUSrc = 1 时，ALU 的第二个运算数来自位扩展之后的立即数。
ALUOp	ALUOp = 00 时，进行加运算，C = A + B； ALUOp = 01 时，进行减运算，C = A - B； ALUOp = 10 时，进行或运算，C = A | B；
MemWr	MemWr = 0 时，DM 不能写入数据； MemWr = 1 时，DM 可以写入数据。
ToReg	ToReg = 00 时，写入寄存器的数据来自 ALU； ToReg = 01 时，写入寄存器的数据来自 DM。 ToReg = 01 时，写入寄存器的数据来自 NPC。

表 3.1.2 非转发暂停控制信号产生的真值表

指令	opcode	funct	branch	Br	RegIn	RegWr	EXTOp	ALUSrc	ALUOp	MemWr	ToReg
addu	000000	100001	0	X	01	1	X	0	00	0	00
add	000000	100000	0	X	01	1	X	0	00	0	00
subu	000000	100011	0	X	01	1	X	0	01	0	00
sub	000000	100010	0	X	01	1	X	0	01	0	00
ori	001101		0	X	00	1	00	1	10	0	00
lw	100011		0	X	00	1	10	1	00	0	01
sw	101011		0	X	X	0	10	1	00	1	X
beq	000100		1	00	X	0	X	0	X	0	X
lui	001111		0	X	00	1	01	1	00	0	00
nop	000000	000000	0	X	00	0	00	0	00	0	00
j	000010		1	01	X	0	X	X	X	0	X
jal	000011		1	01	10	1	X	X	X	0	10
jr	000000	001000	1	10	X	0	X	X	X	0	X
mfc0	010000 00000		0	X	00	1	X	X	X	0	01
			ismfc0 决定存入 GRF 的数据来源是 DM 还是 CP0								
mtc0	010000 00100		0	X	X	0	X	X	X	0	X
			ismtc0 决定数据能否存入 CP0，								
eret	32 位全		1	11	X	0	X	X	X	0	X
			iseret CP0 的 EXL 清零，D 级加 nop								

附表 支持指令集编码与操作

指令	编码形式	操作
addu	000000 rs rt rd 00000 100001	GPR[rd] = GPR[rs] + GPR[rt]
add	000000 rs rt rd 00000 100000	GPR[rd] = GPR[rs] + GPR[rt] 检测溢出
subu	000000 rs rt rd 00000 100011	GPR[rd] = GPR[rs] − GPR[rt]
sub	000000 rs rt rd 00000 100010	GPR[rd] = GPR[rs] − GPR[rt] 检测溢出
ori	100011 base rt offset	GPR[rt] = GPR[rs] OR zero_extend(immediate)

lw	100011 base rt offset	Addr = GPR[base] + sign_ext(offset) GPR[rt] = memory[Addr]
sw	101011 base rt offset	Addr = GPR[base] + sign_ext(offset) memory[Addr] = GPR[rt]

beq	000100 rs rt offset	if (GPR[rs] == GPR[rt])
		PC = PC + 4 + sign_extend(offset‖00)
		else
		PC = PC + 4

lui	001111 00000 rt immediate	GPR[rt] = immediate‖0^16
j	000010 instr_index	PC = PC[31:28] || instr_index || 00
jr	000000 rs 10{0} 5{0} 001000	PC = GPR[rs]
jal	000011 instr_index	PC = PC[31:28] || instr_index || 00 GPR[31] = PC + 4
nop	0x00000000	
mfc0	010000 00000 rt rd 10’b0	GPR[rt] = CP0[rd]
mtc0	010000 00100 rt rd 10’b0	CP0[rd] = GPR[rt]
eret	01000010000000000000000000011000	PC = CP0[epc]

2.AT 编码器

根据指令变量与，产生 Tuse 变量和 Res 编码值（Res 是 Tnew 的变形，用于指定产生新值的功能部件）。

此处将指令分为 Cal_r 类(addu, subu)、Cal_i 类(ori)、Beq 类(beq)、Load 类(lw)、Save 类(sw)分类处理，J 类的三条指令(j, jal, jr)分开处理。

表 4.2.1 指令集的 Tuse

指令	Tuse	
	rs	rt
Cal_r 类	1	1
Cal_i 类	1	
load 类	1	
store 类	1	2

表 3.2.2 指令集的 Tnew

指令	功能部件	Tnew		
		E	M	W
Cal_r 类	ALU	1	0	0
Cal_i 类	ALU	1	0	0
load 类	DM	2	1	0
store 类				

beq	0	0
lui		
nop		
j		
jal		
jr	0	
mfc0		
mtc0		1
eret		

beq				
lui	ALU	1	0	0
nop				
j				
jal	PC	0	0	0
jr				
mfc0	DM	2	1	0
mtc0				
eret				

3.冲突检测器

根据策略矩阵计算分类指令的暂停条件，根据条件按照逻辑取与和或运算。

表 3.3.1 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

表 4.3.2 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

4.转发控制器

根据转发多选器需求和原则产生转发多选器的控制信号，控制信号的优先级由顺序决定，通过三目运算符的比较赋值顺序实现。

转发条件：转发接受级读寄存器编号与被转发级写寄存器编号相同不为 0，转发级寄存器写使能信号为 1。

转发来源：RES 所标记的被转发级指令对应的新值产生部件储存在流水寄存器中的值。

表 3.4.1 转发控制信号

控制信号	取值	功能说明
MF_RD1_D	MF_PC8_E	向 D 级需要 RD1 值的端口转发 PC8_E
	MF_PC8_M	向 D 级需要 RD1 值的端口转发 PC8_M
	MF_AO_M	向 D 级需要 RD1 值的端口转发 AO_M
	MF_RD	D 级需要 RD1 值的端口无需转发
MF_RD2_D	MF_PC8_E	向 D 级需要 RD2 值的端口转发 PC8_E
	MF_PC8_M	向 D 级需要 RD2 值的端口转发 PC8_M
	MF_AO_M	向 D 级需要 RD2 值的端口转发 AO_M
	MF_RD	D 级需要 RD2 值的端口无需转发
MF_RD1_E	MF_PC8_M	向 E 级需要 RD1 值的端口转发 PC8_M
	MF_AO_M	向 E 级需要 RD1 值的端口转发 AO_M
	MF_WD_W	向 E 级需要 RD1 值的端口转发 W 级的 WD 值
	MF_RD	E 级需要 RD1 值的端口无需转发
MF_RD2_E	MF_PC8_M	向 E 级需要 RD2 值的端口转发 PC8_M
	MF_AO_M	向 E 级需要 RD2 值的端口转发 AO_M
	MF_WD_W	向 E 级需要 RD2 值的端口转发 W 级的 WD 值
	MF_RD	E 级需要 RD2 值的端口无需转发
MF_RD2_M	MF_WD_W	向 M 级需要 RD2 值的端口转发 W 级的 WD 值
	MF_RD	M 级需要 RD2 值的端口无需转发

在 datapath 中新增 EPC 的转发：当 E 或 M 级为 MTC0 且写 EPC 寄存器时，从 E 级和 M 级的流水寄存器向 D 级 NPC 的 EPC 端口转发 EPC 值，而其他转发不受影响。

MF_EPC_D	MF_EPC_E	向 D 级 NPC_EPC 端口转发 MF_RD2_E
	MF_EPC_M	向 D 级 NPC_EPC 端口转发 MF_RD2_M
	MF_EPC	D 级 NPC_EPC 端口无需转发

四、测试程序

1.新增指令测试

表 5.1 CPU 测试程序与期望输出（1）

测试程序	期望输出或现象
1 ori $t0,$t0,0x3018	00003000: $ 8 <= 00003018
2 mtc0 $t0,$14	CP0 的 EPC 值变为 0x00003018
3 mfc0 $t1,$14	00003008: $ 9 <= 00003018
4 eret	跳转至第 7 行
5 ori $t0,$t0,0x2333	更改为 nop
6 ori $t1,$t1,0x2333	不执行
7 ori $t1,$t1,0xaaaa	00003018: $ 9 <= 0000baba

2.bridge 与 timer 测试

表 5.2 CPU 测试程序与期望输出（2）

测试程序	期望输出或现象
1 ori $t0,$t0,0x3 #初值	00003000: $ 8 <= 00000003
2 ori $t1,$t1,0x9 #模式 0	00003004: $ 9 <= 00000009
3 sw $t1,0x7f00($0)	timer0 的 PRESET 变为 3
4 sw $t0,0x7f04($0)	timer0 的 CTRL 后四位变为 1001
5-9 nop	timer0 模式 0 计数
10 ori $t1,$t1,0xb #模式 1	00003024: $ 9 <= 0000000b
11 sw $t1,0x7f00($0)	timer0 的 CTRL 后四位变为 1011
12 lw $t2,0x7f08($0)	00003030: $10 <= 00000000
13- nop	timer0 模式 1 计数 a

图 5.1 timer 模式 0 工作方式

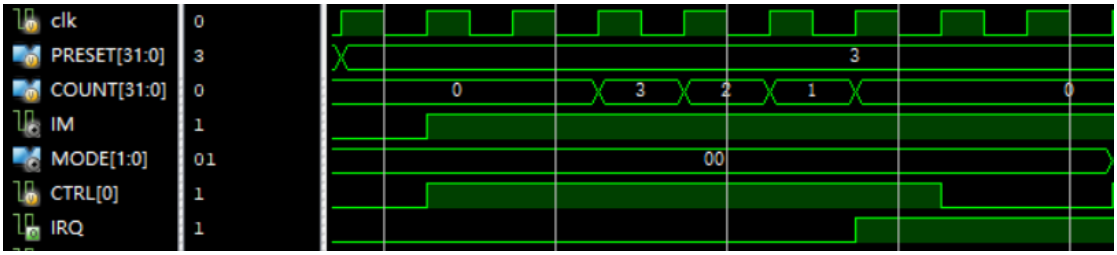
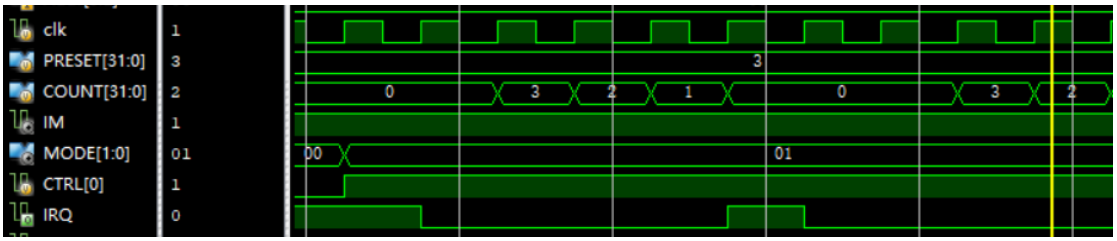


图 5.2 timer 模式 1 工作方式



3.中断异常进入测试

表 5.3 CPU 测试程序与期望输出（3）

序号	测试目的	测试程序	期望输出或现象
1	F 级 RI 异常	ori $t0,0xffff or $t1,$t0,$t0	or 进入 F 级后产生 ExcCode=10，D 级变为 nop，进入 M 级后进入异常处理。
2	F 级 AdEL 异常	ori $t0,0x3009 jr $t0 ... 0x2000,0x5000	jr 跳转的 PC 值进入 F 级后产生 ExcCode=4，指令变为 nop，进入 M 级后进入异常处理。
3	E 级 Ov 异常	lui $t0,0x8000 add $t1,$t0,$t0	add 进入 E 级后产生 ExcCode=12，进入 M 级后进入异常处理。
4	E 级 AdEL 异常	ori $t0,0x8000 lw $t1,0x8000($t0)	
5	E 级 AdES 异常	ori $t0,0x8000 sw $t1,0x8000($t0)	
6	M 级 AdEL 异常	lw $t0,3($0) ... 0x3000,0x7f0e	lw 进入 M 级后产生 ExcCode=4，然后进入异常处理。
7	M 级 AdES 延迟槽指令异常	j hhh sw $t0,3($0) ... 0x3000,0x7f0e	sw 进入 M 级后产生 ExcCode=5，然后进入异常处理，不写 DM。 EPC 存入 j 的地址。

8	中断	ori $t0,0x3
		ori $t1,0x9
		sw $t0,0x7f04($0)
		sw $t1,0x7f00($0)
		ori $t2,0x0401
		mtc0 $t2,$12
			计时器装载 3 开始模式 0 工作，SR 的中断屏蔽 0 位与全局中断使能打开。
			当计时器到 0 时，进入中断。

六、思考题

Q1：我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？

A1：沟通硬件和软件的东西喵？

Q2.1：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。

A2.1：CPU 外部。

Q2.2：BE 部件对所有的外设都是必要的吗？

A2.2：不是。不一定需要截取特定字节。

Q3.1: 请开发一个主程序以及定时器的 exception handler。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。）

A3.1:

```
1  .text
2  initial:
3  ori $t0,0x100          #PRESET 100
4  ori $t1,0x9            #CTRL[3:0] 1 00 1
5  ori $t2,0x401          #SR open IP[0],IE
6  mtc0 $t2,$12           #save SR
7  sw $t0,0x7f04($0)      #save PRESET
8  sw $t1,0x7f00($0)      #save CTRL
9  main:
10 nop
11 nop                    #wait until INT
12 nop
13
14 .ktext 0x00004180
15 ori $t1,0x9            #CTRL[3:0] 1 00 1
16 sw $t1,0x7f00($0)      #save CTRL
17 eret                   #return and count
```

Q3.2：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？

A3.2：鼠标和键盘进行输入时向 CPU 发出中断信号，并产生对应操作的信号，在中断中传给 CPU 进行处理，处理结束后结束中断。