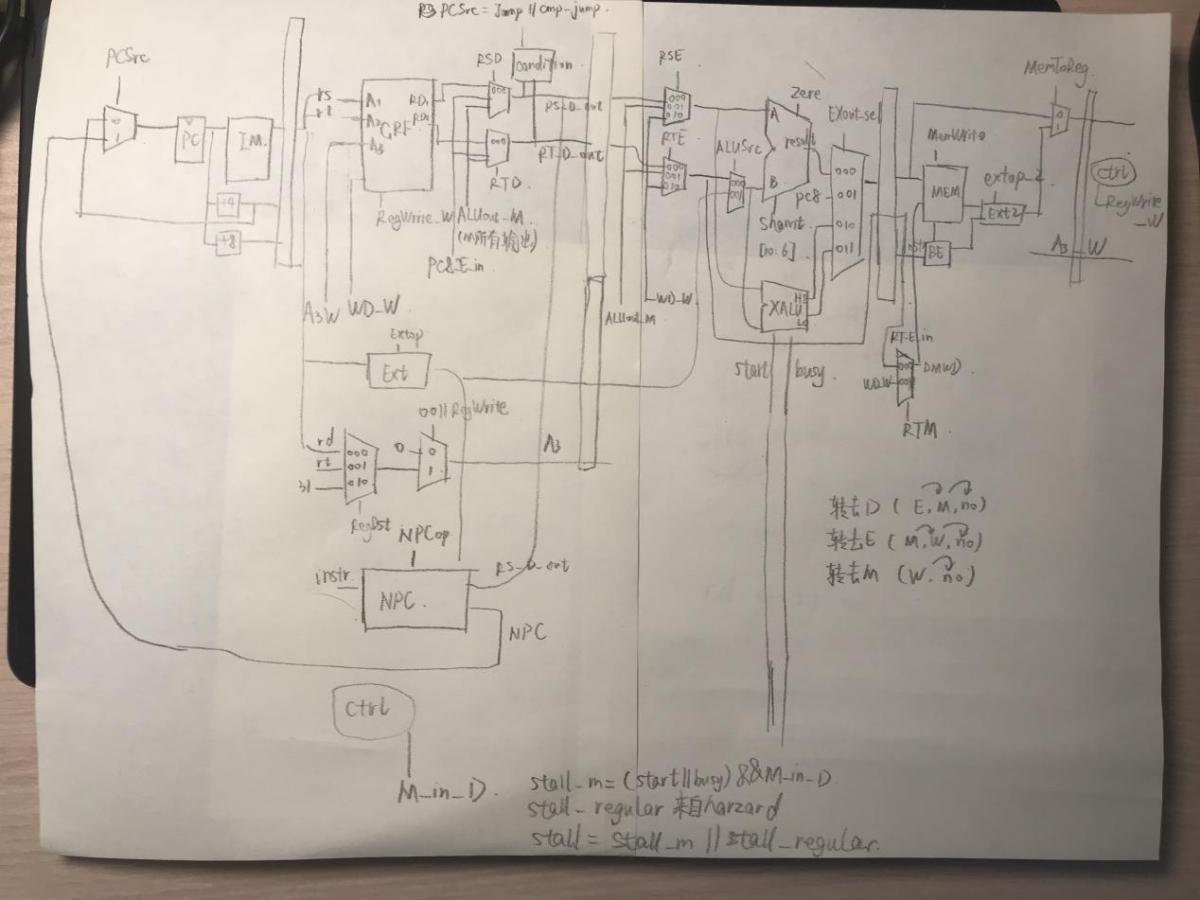
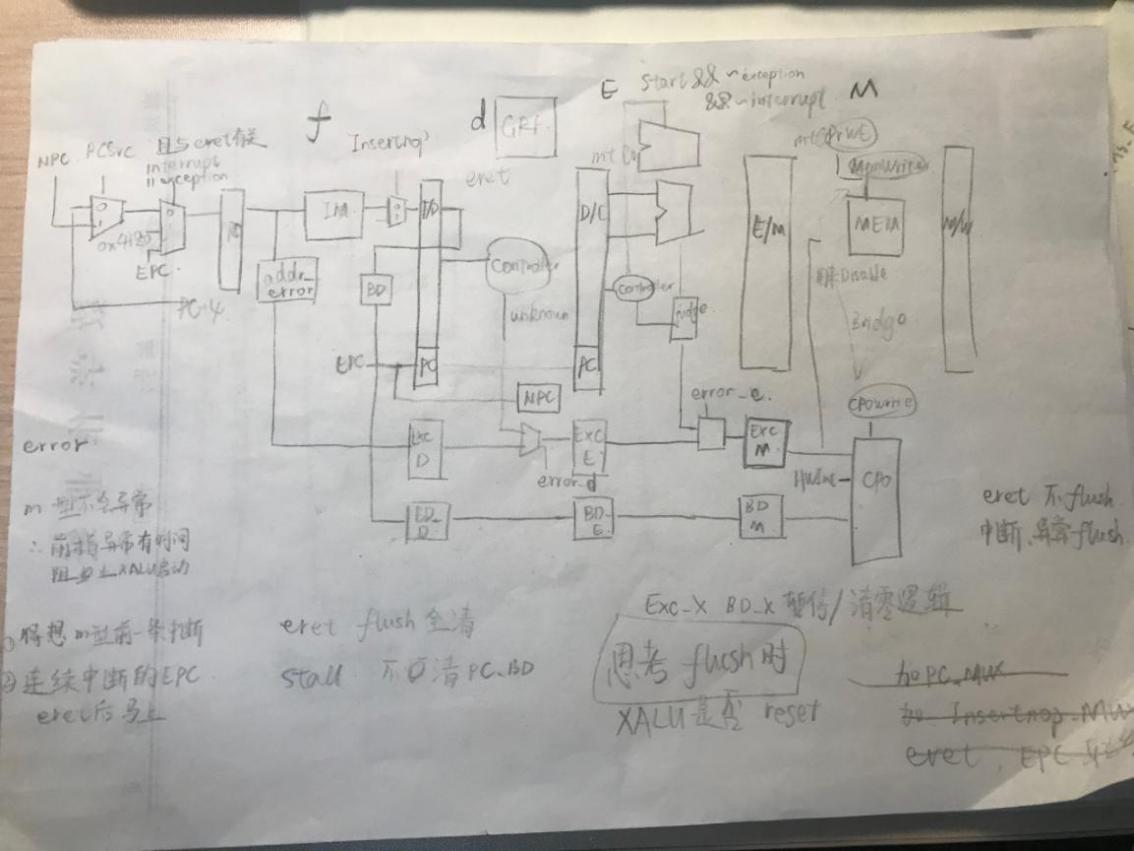
MIPS流水线处理器

一、通路构造





1. 模块说明
2. PC（程序计数器）

用于输出当前指令的PC值，和保存下一条指令的PC值。该模块由一个32位寄存器构成。

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| NextPC[31:0] | I | 下一条指令的PC值 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：有效  0：无效 |
| PC[31:0] | O | 当前指令PC值 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出PC | 在clk上跳沿输出当前指令的PC值 |
| 2 | 复位 | 当复位信号有效时，PC被设置为起始地址0x00003000 |

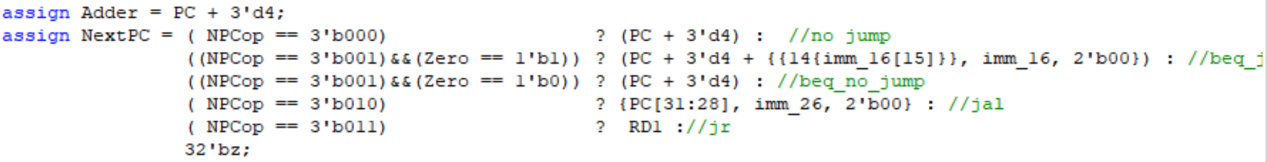
1. NextPC

用于计算下一条指令的PC值

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC [31:0] | I | 当前指令PC值 |
| Imm\_16[15:0] | I | 16位立即数 |
| Imm\_26[25:0] | I | 26位立即数 |
| NPCop [2:0] | I | NPC操作指令 |
| RD1 [31:0] | I | 当前RD1 |
| zero | I | ALU计算结果为0标志  1：ALU的两个输入相等  0：ALU的两个输入不等 |
| Adder[31:0] | O | PC + 4 |
| NextPC | O | 下一条指令的PC值 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出PC | 输出下一条指令的PC值 |
| 2 | PC + 4 | 输出当前指令的PC + 4 |



1. IM（指令存储器）

IM容量为32bit×1024，实际地址宽度为10位，从而将地址的低10位（2~11位）连接到IM选择地址端口。

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[13:2] | I | 当前指令的PC |
| Opcode [5:0] | O | 6位opcode信号 |
| Func [5:0] | O | 6位Func信号 |
| rs [5:0] | O | rs寄存器编号 |
| rt [5:0] | O | rt寄存器编号 |
| rd [5:0] | O | rd寄存器编号 |
| imm\_16[15:0] | O | 16位立即数 |
| Imm\_26[25:0] | O | 26位立即数 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出指令 | 根据当前PC值输出所对应的指令 |

1. GRF（通用寄存器组）

由32个32位寄存器构成，其中$0始终保持为0

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：有效  0：无效 |
| PC [31:0] | I | 当前指令的PC值 |
| RegWrite | I | 读写控制信号  1：写操作  0：读操作 |
| RA1 [4:0] | I | 读寄存器1的地址 |
| RA2 [4:0] | I | 读寄存器2的地址 |
| WA [4:0] | I | 写寄存器的地址 |
| WD [31:0] | I | 向写寄存器中写入的值 |
| RD1 [31:0] | O | RA1所对应的寄存器的值 |
| RD2 [31:0] | O | RA2所对应的寄存器的值 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有寄存器的值被设置为0 |
| 2 | 写寄存器 | 根据输入的写寄存器地址，把输入的数据写入写寄存器中 |
| 3 | 读寄存器 | 根据输入的读寄存器地址，将数据读出 |

1. ALU（算术逻辑单元）

提供32位加、减、或运算

可以不支持溢出

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A [31:0] | I | ALU32位输入数据A |
| B [31:0] | I | ALU32位输入数据B |
| ALUop[1:0] | I | ALU功能选择信号  00:加法  01:减法  10:或运算 |
| Result[31:0] | O | 32位计算结果 |
| Zero | O | A == B |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加 | A + B |
| 2 | 减 | A - B |
| 3 | 或 | A | B |

1. DM（数据存储器）

 DM容量为32bit×1024，其起始地址为0x00003000

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：有效  0：无效 |
| MemWrite | I | 读写控制信号  1：写操作 |
| PC | I | 对应指令的PC值 |
| Address [31:0] | I | 所要进行操作的地址 |
| WD [31:0] | I | 写入数据的输入 |
| RD [31:0] | O | 读取数据的输出 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有数据被设置为0x00000000 |
| 2 | 写操作 | 根据输入address，把输入的数据写入 |
| 3 | 读操作 | 根据输入address，将其中的数据读出 |

1. EXT（位扩展器）

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm\_16 [15:0] | I | 16位立即数 |
| Extop[1:0] | I | 位扩展选择信号  00：无符号扩展  01：有符号扩展  10：扩展至[31:16]位，低16位补0 |
| out [31:0] | O | 位扩展后的32位输出 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 高位补0 | 高16位补0 |
| 2 | 低位补0 | 低16位补0 |
| 3 | 符号扩展 | 若符号位为0，则高位补0  若符号位为1，则高位补1 |

1. 控制器（Controller）

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Opcode[5:0] | I | Opcode |
| Func[5:0] | I | Func |
| RegDst[2:0] | O | WA输入的选择信号 |
| NPCop [2:0] | O | NPC模块的行为控制信号 |
| MemToReg[2:0] | O | WD输入的选择信号 |
| RegWrite | O | GRF写使能 |
| MemWrite | O | DM写使能 |
| ALUSrc [2:0] | O | ALU的B输入的控制信号 |
| Extop[1:0] | O | 位扩展控制信号 |
| ALUop[1:0] | O | ALU行为控制信号 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | NPCop | MemToReg | RegWrite | MemWrite | ALUSrc | Extop | ALUop |
| addu | 000 | 000 | 000 | 1 | 0 | 000 | X | 00 |
| subu | 000 | 000 | 000 | 1 | 0 | 000 | X | 01 |
| ori | 001 | 000 | 000 | 1 | 0 | 001 | 00 | 10 |
| lw | 001 | 000 | 001 | 1 | 0 | 001 | 01 | 00 |
| sw | X | 000 | X | 0 | 1 | 001 | 01 | 00 |
| beq | X | 001 | X | 0 | 0 | 000 | 01 | X |
| lui | 001 | 000 | 000 | 1 | 0 | 001 | 10 | 00 |
| jal | 010 | 010 | 010 | 1 | 0 | X | X | X |
| jr | X | 011 | X | 0 | 0 | X | X | X |
| nop | / | / | / | / | / | / | / | / |

1. Bridge

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PrAddr[31:0] | I | 写入数据的地址 |
| PrWE | I | 写入数据的使能信号 |
| PrWD[31:0] | I | 写入的数据 |
| IQR0 | I | 来自timer0的中断请求 |
| IQR1 | I | 来自timer1的中断请求 |
| interrupt | I | 外部中断信号 |
| DEV0\_RD[31:0] | I | 从timer0读出的数据 |
| DEV1\_RD[31:0] | I | 从timer1读出的数据 |
| PrRD | O | 从外设读入的数据 |
| HWInt | O | 六个中断请求 |
| Addr | O | 写入外设的地址 |
| WE0 | O | Timer0的写使能 |
| WE1 | O | Timer1的写使能 |
| DEV\_WD[31:0] | O | 向外设写入的数据 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 挂接外设 | 将timer0、timer1挂接在bridge上，并在CPU的控制下读写 |
| 2 | 内外部通信 | 从外部向内部传入中断信号 |

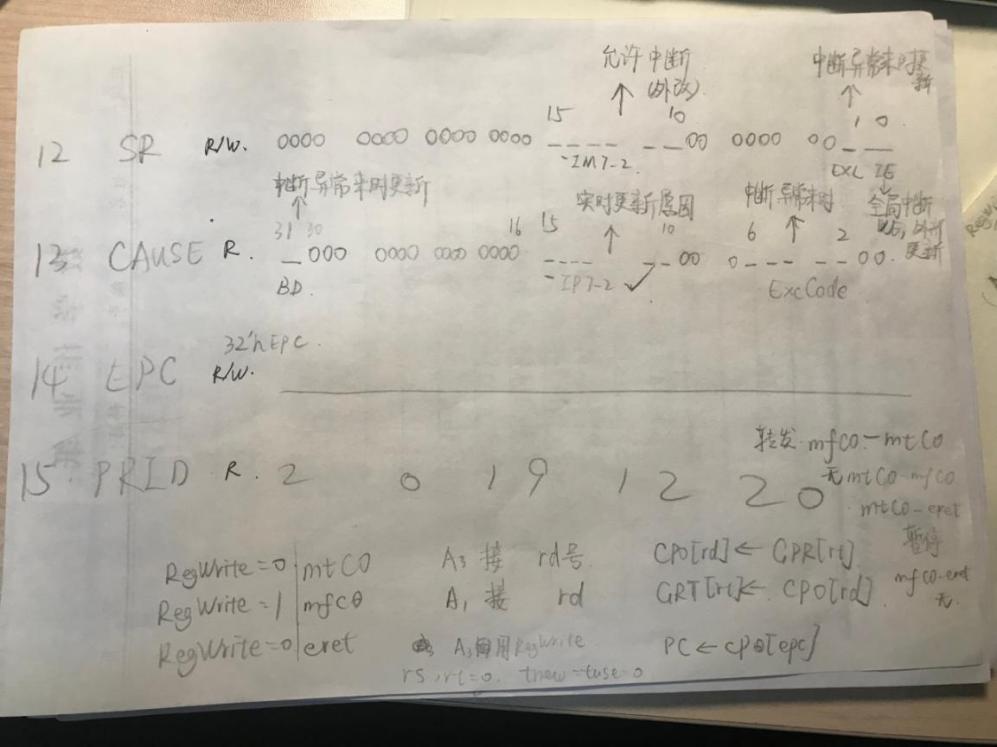
1. CP0寄存器

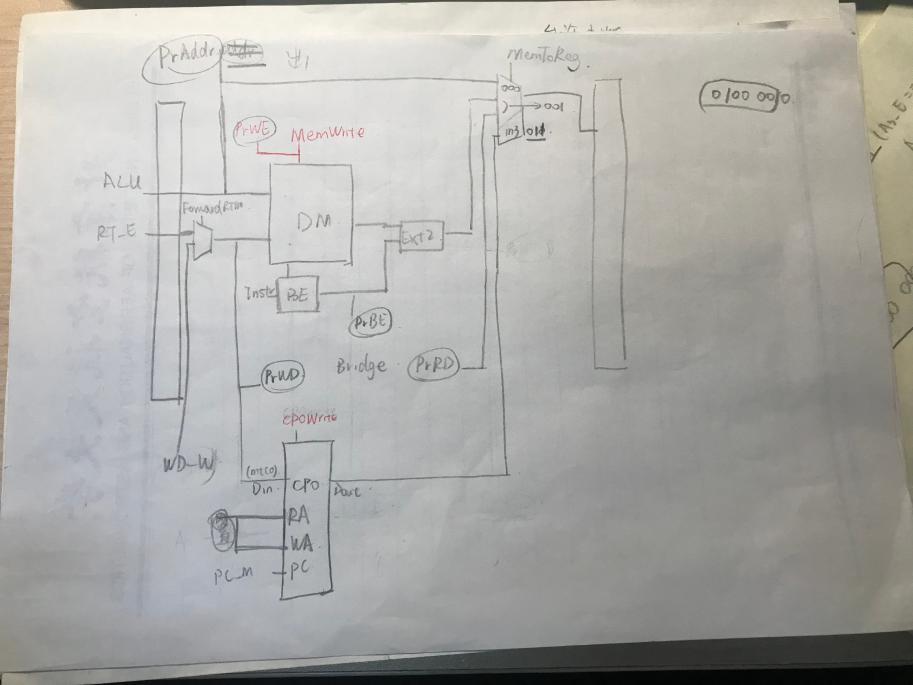
模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| Cp0write | I | CP0寄存器写使能 |
| RA[4:0] | I | 读取数据的地址 |
| WA[4:0] | I | 写入数据的地址 |
| Din[31:0] | I | 写入的数据 |
| PC[31:0] | I | 当前M级指令受害的EPC |
| ExcCode\_in[4:0] | I | 当前M级指令的ExcCode |
| HWInt[5:0] | I | 六个中断源 |
| BD\_in | I | 当前M级指令是否为延时槽指令 |
| Eret | I | 当前M级为一条eret指令 |
| Interrupt | O | CPU进入中断处理 |
| Exception | O | CPU进入异常处理 |
| EPC[31:0] | O | 若结束中断\异常处理时应该返回的PC |
| Dout[31:0] | O | CP0的输出数据 |

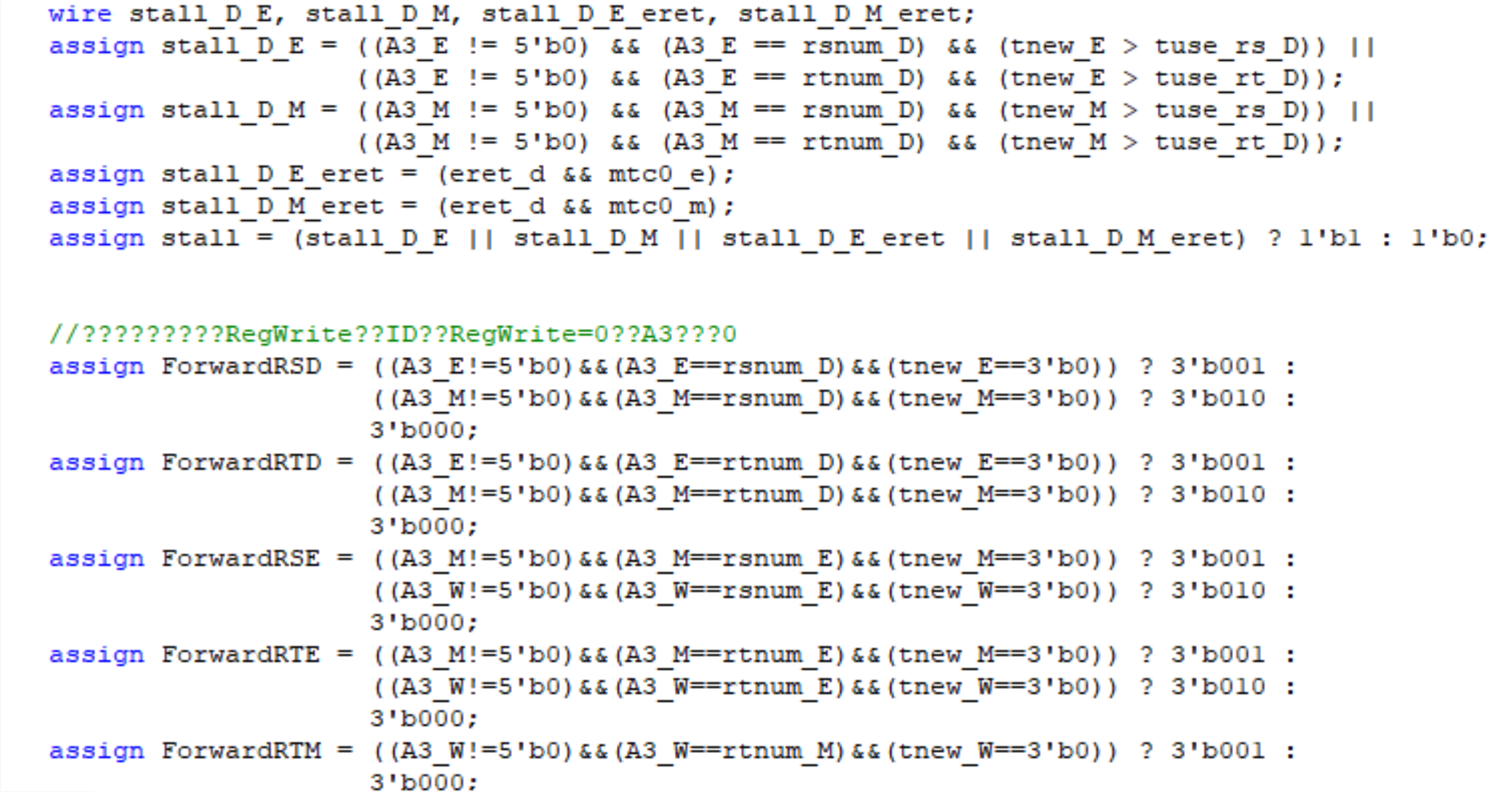
功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 记录错因 | CP0在每个中断和异常到来时，记录ExcCode和IP |
| 2 | 触发中断处理 | 根据IP、IM、IE、EXL决定当前时刻是否应该中断 |
| 3 | 触发异常处理 | 根据IP、IM、EXL决定当前时刻是否应该处理异常 |
| 4 | 屏蔽异常及中断 | 根据IP、IM、IE、EXL决定当前时刻是否应该处理中断和异常 |





1. 暂停和转发



1. 测试程序

**1.timer**

ori $t1, $zero, 0x00000007

ori $t3, $zero, 0x00000009

ori $t2, $zero, 0x0000fc01

mtc0 $t2, $12

sw $t1, 0x7f04($zero)

sw $t3, 0x7f00($zero)

sw $t1, 4($zero)

sw $t1, 8($zero)

sw $t1, 0xc($zero)

sw $t1, 0x10($zero)

sw $t1, 0x14($zero)

sw $t1, 0x18($zero)

sw $t1, 0x1c($zero)

sw $t1, 0x20($zero)

sw $t1, 0x24($zero)

sw $t1, 0x28($zero)

sw $t1, 0x2c($zero)

#sw $t1, 0x7f08($zero)

lw $t1, 0x7f00($zero)

lw $t1, 0x7f04($zero)

lw $t1, 0x7f08($zero)

sh $t1, 0($zero)

lh $t2, 0($zero)

sh $t1, 1($zero)

lh $t2, 1($zero)

lhu $t2, 1($zero)

sb $t1, 3098($zero)

lbu $t2, 3097($zero)

exit:

beq $zero, $zero, exit

Nop

**2.mult\_BD\_INTERRUPT\_EXCEPTION**

.text

ori $2, $0, 0x1001

mtc0 $2, $12

ori $t1, $zero, 0x7fffffff

add $t2, $t1, $t1

div $t1, $t1

add $t2, $t1, $t1

mfhi $t3

mflo $t4

sh $t1, 1($zero)

lw $t2, 0($zero)

beq $zero, $zero, next

add $t2, $t1, $t1

next:

sub $t2, $t1, $t1

**3.mult\_INTERRUPTED\_RESTORE**

.text

ori $t0, $0, 0xfc01

mtc0 $t0, $12

ori $t1, $zero, 0x3546

mult $t1, $t1

mfhi $t1

mflo $t2

exit:

beq $zero, $zero, exit

Nop

**4.BD\_stall**

ori $t1, $zero, 0x7ffffff

sw $t1 ,0($zero)

lw $t1, 0($zero)

beq $zero, $t1, 1

add $t2, $t1, $t1

1:

beq $zero, $zero, 2

add $t2, $t1, $t1

2:

add $t1, $t1, $t1

or $t3, $t1, $t1

exit:

beq $zero, $zero, exit

nop

**5.synthetic**

.ktext 0x4180

lui $29, 0xffff #进入异常处理

mfc0 $28, $14

addi $28, $28, 4

mtc0 $28, $14

eret #修改PC为下一条指令

lui $29, 0x1234 #不应该触发

.text

li $2, 0x7fffffff

add $2, $2, $2

addi $2, $2, 0x7fffffff

li $3, 0x80000001

sub $3, $3, $2

nop#OV

sw $2, ($0)

sw $2, 0x3fff($0)

sw $2, 1($0)

sh $2, 3($0)

sh $2, 0x7f00($0)

sb $2, 0x7f00($0)

sw $2, 0x7f08($0) #地址

sw $2, 0x7fff($2)

#ADES

lw $2, 1($0)

lh $2, 1($0)

lhu $2, 3($0)

lw $2, 0x3fff($0)

lw $2, 0x7fff($2)

lb $2, 0x7f00($0)

#ADEL

li $2, 0x3001

jr $2

nop

li $2, 0x5120

jr $2

nop

#ADEL

#RI

1. 思考题
2. 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”？

指令集体系结构是软件和硬件的接口。指令集体系结构是指软件能够感知到的部分，也称软件可见部分。同时，指令集体系又是“直接”控制硬件进行工作的体系结构。由此，指令集系统被称为软件和硬件的接口。

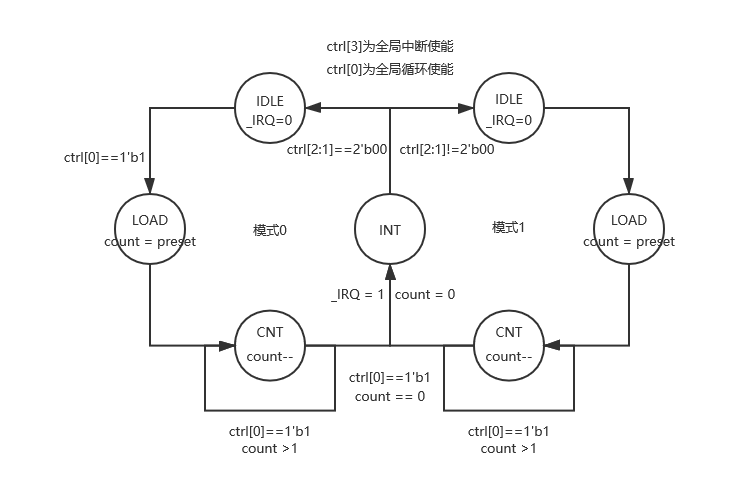
1. 在我们涉及到流水线中，DM处于CPU内部，请你考虑现代计算机中它的位置应该在何处？

应该处在CPU外部，与其他外设一起挂接在Bridge上。

1. BE部件对所有的外设都是必要的吗？

不是对所有的外设都有必要。例如，我们规定timer只能按字存取，因此不需要BE信号来决定哪每个字节是否有效。

1. 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

如果设置为模式1，且timer在允许中断的状态下，则timer按照preset中的数字为周期数，稳定地产生周期性的中断请求信号。

如果设置为模式0，且timer在允许中断的状态下，则timer在发送完当前循环中的中断信号后，需要程序“手动”置位ctrl[0]为1后才可以就下一个记数中断周期。

1. 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将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）

.ktext

ori $t1, $zero, 0x00000009

sw $t1, 0x7f00($zero) #set CTRL

.text

#enable timer interrupt

ori $t1, $zero, 0x0000fc01

mtc0 $t1, $12

#initialize Timer0

#mode 0

ori $t2, $zero, 0x00000005

sw $t2, 0x7f04($zero) #set preset

ori $t1, $zero, 0x00000009

sw $t1, 0x7f00($zero) #set CTRL

1. 请查阅相关资料，说明鼠标和键盘的输入信号的如何被CPU知晓的？

CPU通过轮询知晓外部的输入信号。通向设备的通信通路上通常有2个寄存器，分别为控制寄存器和数据寄存器。处理器会根据每种外设的传输速率，周期性的查询控制寄存器，等待设备置位。之后，处理器会通过load和save指令，向从timer中获取及存入数据一样访问设局寄存器，从而得到外设输入的数据，或者向外设发送数据。