流水线 32 位 CPU——Verilog 实现

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的流水线 MIPS - CPU，支持的指令集包含 MIPS-C4 指令。为了实现这些功能，整个系统对 CPU 进行了修改，增加了 CP0 部件。整体包括 Cpu，Bridge，2 个 timer。

（二）关键模块定义

1. Bridge

信号名	方向	说明
addr_Cpu	I	Cpu 到 Bridge 的地址
dIn_Cpu	I	Cpu 到 Bridge 的数据
dIn_Dev0	I	设备 0 的数据
dIn_Dev1	I	设备 1 的数据
ifWr_Cpu	I	Cpu 是否要向外接设备写数据
ifWrDev0	O	是否要写设备 0
ifWrDev1	O	是否要写设备 1
addr_Dev	O	Cpu 要操作外接设备的地址
dOut_Dev	O	Cpu 向设备输出的数据
dOut_Cpu	O	设备向 Cpu 输入的数据

如果 Cpu 要向外部写数据，并且 Cpu 提供的地址在某个设备的地址范围，就把该设备的写控制置 1。如果 Cpu 的地址在某个设备的范围，就把该设备的数据提供给 Cpu。

2. Device(Timer)

信号名	方向	说明
clk	I	时钟信号
reset	I	重置信号
Addr	I	输入的地址
WE	I	是否要写这个设备
Din	I	输入的数据
Dout	O	输出的数据
IRQ	O	模拟输出中断信号

用于产生中断。

3.Cpu

信号名	方向	说明
clk	I	时钟信号
reset	I	重置信号
dIn_Bridge	I	从 Bridge 来的数据
hwInt_Outside	I	外来的中断信号
dOut_Bridge	O	输出到 Bridge 的数据
addr_Bridge	O	输出到 Bridge 的地址
ifWr_Bridge	O	是否要写外接设备
globalAddr	O	全局 Pc

相比于没有中断异常，增加指令：eret，mfc0，mtc0.

在 M 级增加模块 Cp0.

外来中断信号输入到 M 级的 Cp0.

输出的数据为本来给 Grf 的转发后的数据.

输出的地址为 Alu 计算出的本来给 Dm 用的地址.

4.Cp0

信号名	方向	说明
clk	I	时钟信号
reset	I	重置信号
wEn	I	是否要写 Cp0（ifMtc0）
exlSet	I	是否要把 exl 置 1（Mem 级有异常）
exlClr	I	是否要把 exl 清 0（Eret）
ifBd	I	是否是延迟槽指令
hwInt	I	外来中断
excCode	I	异常码
rA	I	读地址
wA	I	写地址
dIn	I	输入的数据
pc	I	可能要保存的 pc
epc	O	保存的 pc
dOut	O	输出的数据
ifExcOrInt	O	是否有中断或异常

按规范定义各个寄存器.

assign ifExcOrInt = (!exlReg) & (((ieReg) & (|(imReg & hwInt))) || exlSet);

如果有中断或异常，保存 pc：

epcReg <= (ifBd)? ({pc[31:2], 2'b0} - 4) : {pc[31:2], 2'b0};

开始的时候 imReg 所有位都置 1，ieReg 置 1.

优先级：reset、一直保存中断、exlClr、ifExcOrInt、wEn

（三）重要机制实现方法

1.处理中断和异常

所有的流水线寄存器增加 flush 信号，接收 Mem 级流出的 ifEret 和 IfExcOrInt 信号，如果有其中一个，就直接按 reset 的操作全部重置。

在 Mem 判断中断异常以后，pc 跳异常处理程序之前，不能有任何指令执

行完成。最早的就是对 Ex 级的 Hilo 寄存器进行操作。所以 Ex 接收 Mem 级流出的 ifEret 和 IfExcOrInt 信号，如果有其中一个，就不能写入 Hilo 寄存器。

每一级输出 excCode 信号并流水。如果前级有就用前级的，前级为零就用本级的。最后流水到 Mem 级，结合 Mem 级自己的进行判断，输入给 Cp0.

如果 Id 级有分支信号，就把 Bd 信号延迟一级，给 IfId 寄存器，并往下流水，不用经过级，直接从流水线寄存器流水。

2.全局信号

ifBd 和 globalPc 是全局信号，也是给 Cp0 的信号。从 Mem 级的输入往前推，如果某一级 pc 不为 0，或者异常码为 4（在 Mem 之前该情况只可能是取的 pc 没对齐，这是特殊情况），就用这一级的 pc 和 ifBd.

（一）异常测试样例

```
.text
    initialize:
    ori $21, $0, 1
    ori $22, $0, 2
    ori $23, $0, 3
    ori $24, $0, 4
    Ov:
        #solve_1--------------------#
        ori $10, 1
        lui $1, 0x7fff
        lui $2, 0x7fed
        add $3, $1, $2              #error
        lui $1, 0x7fff
        ori $2, $0, 0xffff
        add $1, $1, $2
        ori $2, $0, 2
        add $3, $1, $2              #error
```

```
    lui $1, 0x7fff
    lui $2, 0xffff
    sub $3, $1, $2              #error
    ################################
AdEL:
    #solve_2--------------------#
    ori $10, $0, 2
    #未对齐
    ori $1, $0, 1
    lw $2, 0($1)                #error
    ori $1, $0, 1
    lh $2, 0($1)                #error
    ori $1, $0, 1
    lhu $2, 0($1)           #error
    #取 Timer
    ori $1, $0, 0x7f00
    lh $2, 0($1)                #error
    ori $1, $0, 0x7f00
    lhu $2, 0($1)           #error
    ori $1, $0, 0x7f00
    lb $2, 0($1)                #error
    ori $1, $0, 0x7f00
    lbu $2, 0($1)           #error
    #地址溢出
    addi $1, $0, -1
    lb $1, 0($1)                #error
    addi $1, $0, -1
    lbu $1, 0($1)           #error
    addi $1, $0, -2
    lh $1, 0($1)                #error
```

```
addi $1, $0, -2
lhu $1, 0($1)          #error
addi $1, $0, -4
lw $1, 0($1)           #error
#地址不在范围
ori $1, $0, 0x5000
lb $2, 0($1)           #error
ori $1, $0, 0x5000
lbu $2, 0($1)          #error
ori $1, $0, 0x5000
lh $2, 0($1)           #error
ori $1, $0, 0x5000
lhu $2, 0($1)          #error
ori $1, $0, 0x5000
lw $2, 0($1)           #error
##############################
AdEs:
#solve_2--------------------#
ori $10, $0, 2
#未对齐
ori $1, $0, 1
sw $2, 0($1)           #error
ori $1, $0, 1
sh $2, 0($1)           #error
#存 Timer
ori $1, $0, 0x7f00
sh $2, 0($1)           #error
ori $1, $0, 0x7f00
sb $2, 0($1)           #error
```

```
        #地址溢出
        addi $1, $0, -1
        sb $1, 0($1)            #error
        addi $1, $0, -2
        sh $1, 0($1)            #error
        addi $1, $0, -4
        sw $1, 0($1)            #error
        #存 Count
        addi $1, $0, 0x7f08
        sw $1, 0($1)            #error
        #地址不在范围
        ori $1, $0, 0x5000
        sb $2, 0($1)            #error
        ori $1, $0, 0x5000
        sh $2, 0($1)            #error
        ori $1, $0, 0x5000
        sw $2, 0($1)            #error
        ##############################

.ktext 0x4180
    mfc0 $4, $12
    mfc0 $4, $13
    mfc0 $4, $14
    beq $10, $21, solve_1
    nop
    beq $10, $22, solve_2
    nop
    solve_1:
        add $2, $0, $0
        eret
```

```
solve_2:
    ori $1, $0, 0
    eret
```

（二）中断测试样例

```
.text
    #方式 0
    ori $1, $0, 9           #ctrl
    ori $2, $0, 5           #present
    ori $3, $0, 0x7f00      #地址
    sw $1, 0($3)
    sw $2, 4($3)
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    #方式 1
    ori $1, $0, 11          #ctrl
    ori $2, $0, 5           #present
    ori $3, $0, 0x7f00      #地址
    sw $1, 0($3)
    sw $2, 4($3)
```

```
.ktext 0x4180
    mfc0 $10, $12
    mfc0 $10, $13
    mfc0 $10, $14
    ori $4, $0, 0       #ctrl
    ori $5, $0, 0x7f00          #地址
    sw $4, 0($5)
    nop
    nop
    nop
    eret
```

三、思考题

（一）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”？（Tips：什么是接口？和我们到现在为止所学的有什么联系？）

类似于规范硬件和软件行为的协议。硬件给出硬件如何操作和信号意义的说明书，软件按照说明来操作。

（二）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。

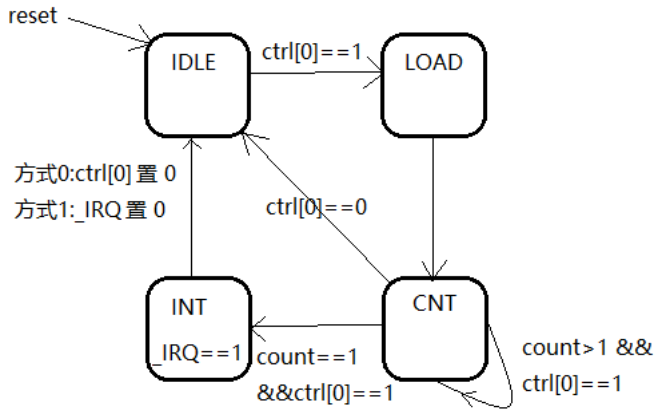
在 Cpu 外，用总线和 Cpu 连接。

（三）BE 部件对所有的外设都是必要的吗？

不一定，如果只是按字访问，就不需要。

（四）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。

只有从 INT 到 IDLE 的一步不一样，方式 0 是把 ctrl[0]置 0，方式 1 是把_IRQ 置 0。这一步中方式 0 的中断会延续，方式 1 由于_IRQ 为零了，中断结束。



（五）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 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。

```
.text
addi $1, $0, 0x7f00
addi $2, $0, 100
addi $3, $0, 9
```

```
addi $4, $0, 0x401
mtc0 $4, $12
sw $2, 4($1)
sw $3, 0($1)
bTo:
beq $0, $0, bTo
lw $4, 8($1)

.ktext 0x4180
sw $3, 0($1)
eret
```

（六）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？

如果按鼠标和键盘，就会发出一个中断信号，让 CPU 处理这个中断。