计算机组成原理实验报告

18375200 刘裕炜

一、CPU设计方案综述

（一）总体设计概述

本CPU为Verilog实现的五级流水线MIPS - CPU，支持的指令集为MIPS-C3。为了实现这些功能，CPU主要包含了IFU、GRF、ALU、DM、指令译码等模块。

（二）关键模块定义

1、GRF

表1 GRF模块接口

信号名	方向	描述
RAddr0[4:0]	I	要读取寄存器的第一个地址
RAddr1[4:0]	I	要读取寄存器的第二个地址
WAddr[4:0]	I	写入地址
WriteData[31:0]	I	写入数据
WritePC[31:0]	I	执行写入的指令地址
clk	I	时钟
reset	I	同步复位信号
RData0[31:0]	O	第一个地址寄存器的值
RData1[31:0]	O	第二个地址寄存器的值

表2 GRF功能定义

序号	功能名称	功能描述
1	读寄存器	RData0输出RAddr0地址所寻址的寄存器，RData1输出RAddr1地址所寻址的寄存器
2	写寄存器	RegWrite有效且reset不为1时，在时钟上升沿将WD的数据写入A3地址所寻址的寄存器
3	同步复位	clk上升沿时若reset信号有效，将32个GPR单元清零

2、IM

表3 IM模块接口

信号名	方向	描述
OpAddr[31:0]	I	指令地址
OpCode[31:0]	O	指令数

3、ALU

表4 ALU模块接口

信号名	方向	描述
In0[31:0]	I	第一个运算数
In1[31:0]	I	第二个运算数
ALUOp[2:0]	I	ALU控制信号
Res[31:0]	O	运算结果

表5 ALU功能定义

序号	功能名称	功能描述
1	加法	Res输出In0+In1的值
2	减法	Res输出In0-In1的值
3	按位或	Res输出In0和In1按位或的值
4	判断相等	若In0和In1相等，Zero输出1，否则输出0

4、DM

表6 DM模块接口

信号名	方向	描述
Addr[31:0]	I	操作地址
WData[31:0]	I	写入数据
MemWrite	I	写入使能
clk	I	时钟
reset	I	异步复位信号
WritePC[31:0]	I	执行写入的指令地址
RData[31:0]	O	操作地址对应内存单元存储的值

5、NPC

表7 NPC模块接口

信号名	方向	描述
PC[31:0]	I	当前PC值
branch_addr[31:0]	I	分支地址
jump_addr[31:0]	I	跳转地址
branch	I	branch信号
jump	I	jump信号
Stall	I	阻塞信号
NextPC[31:0]	O	下一条PC值
PC_4[31:0]	O	PC+4

6、指令译码器

指令译码器位于ID模块，用于将指令码译码，得到具体的指令类型，然后将指令类型流水至各级。

表8 控制器模块接口

信号名	方向	描述
Instr[31:0]	I	指令码
InstrType[59:0]	O	指令类型（独热码）

（三）关键功能实现

1、GRF的同时读写

在GRF中设置了内部转发，当同时对同一个寄存器进行读写时，将待写入数据直接交给读出端口。

2、分支和跳转

（1）分支

beq指令的相等判断前移至ID级，取出数据后立即判断得到是否转发的信号，并将转发信号和转发地址不经流水线寄存器直接连回NPC，这样可以做到只留一个延迟槽。

（2）跳转

j和jr指令在ID级译码后直接将跳转地址和信号连回NPC。

jal指令在ID级译码后将跳转地址和信号连回NPC，将PC+8的值继续流水，在EX级合并入R型计算指令的数据通路，视作在EX级产生结果。

3、冲突处理策略

T_{use} ： 该寄存器自当前时刻起算，再过多少个时钟周期，它的值需要被准备好供使用。

T_{new} ： 自当前时刻起算，再过多少个时钟周期，准备存入该寄存器的结果可以从某个流水线寄存器中取出。

采取计算 $T*{new}$和$T*{use}$的方法来确定是否转发或阻塞。为达到转发和阻塞效果，需要将待读取地址Rs、Rt，待写入地址Rt，Rd随指令一起流水。且在ID级解码产生信号类型时也应该一并产生各寄存器的 $T*{new}$ 和 $T*{use}$ ，在ID级判断是否阻塞，当阻塞解除后，将 $T*{new}$ 和 $T*{use}$ 逐级流水，并在进入新一级流水时将非0的值减1。

指令类型	T_{use}	$T_{new}\ in\ ID$	$T_{new}\ in\ EX$	$T_{new}\ in\ Mem$
R型计算指令	1	2	1	0
I型计算指令	1	2	1	0
load.rs (addr)	1	\	\	\
load.rt	\	3	2	1
store.rs (addr)	1	\	\	\
store.rt (value)	2	\	\	\
lui	\	1	0	0
jal	\	1	0	0
jr	0	\	\	\
jalr	0	1	0	0
两操作数branch	0	\	\	\
单操作数branch	0	\	\	\
mthi，mtlo	1	\	\	\
mult和div类	1	\	\	\
mflo，mfhi	\	2	1	0

（1）lui和jal指令

lui和jal指令仅对立即数处理，因此在ID阶段就可以得到结果，在EX阶段将其并入R型指令的数据通路。

（2）阻塞方法

对同一个寄存器，存在某级$T*{new}>T*{use}$ 时，说明数据不能在使用的时刻前产生，需要阻塞。阻塞时，冻结IF/ID和PC寄存器，同步清零ID/EX寄存器，并更新前一指令的 T_{new} 值。

当乘除模块的Start或Busy信号有效时，检查ID级是否为乘除模块相关指令，若是则阻塞至Busy信号解除为止。

（3）转发方法

在排除阻塞情况后，对每个支持转发的元件数据输入的端口，其MUX的控制信号中，0表示原先数据通路来源的数据，其它接口为相应转发来源，转发信号通过比对需求数据的寄存器编号和后续流水线中最近的命中的寄存器编号决定是否执行转发。

转发节点如下：

- 转发供给：
  - ID/EX寄存器中ResFromID_ID_to_EX，对应于在ID阶段产生的结果；
  - EX/Mem寄存器中的ALUOut_EX_to_Mem，对应于在EX及其之前阶段产生的结果；
  - Mem/WB寄存器后，DM数据扩展单元后的RegWriteData_Mem_to_WB，对应于在Mem及其之前阶段产生的结果。
- 转发需求：
  - ID级中GRF寄存器出口，与GRF两个读出数据合并，进行选择；
  - EX级中ALU的两个数据入口；
  - EX级中准备带入Mem级的写入数据；
  - Mem级中DM的写入数据入口。

具体转发方案：

- 每级流水中传下来的Rs，Rt的地址，若非0，则为需要读取且需要注意冲突的寄存器地址。每级流水中的RegWriteAddr若非0则为最终要写的寄存器地址。
- ID级中GRF寄存器出口仅在当指令在ID级立即需要后面级流水已经计算结束但还没回写的数据时进行转发，因此只需连接ResFromID_ID_to_EX和ALUOut_EX_to_Mem两个来源，并在转发控制单元内对这两个来源的寄存器号进行比对即可。对于Mem/WB寄存器的来源，由于GRF自身有内部转发机制因此不用单独转发。
- EX级中ALU入口对应于需要在ALU利用存在冲突的寄存器内容执行计算的情况。需要连接ALUOut_EX_to_Mem和RegWriteData_Mem_to_WB两个来源。
  - 注意ALUIn1处立即数优先级应当比转发高，即如果指令需要立即数参与运算，则应当忽略转发的结果。
- 对于DM的写入数据需要读取的寄存器：
  - 如果其上一次被写的指令与当前store类指令间隔有两条及以上，则在store指令的ID阶段，之前对其写入的指令最迟已经走到WB阶段，此时要么是无冒险，要么是同时读写，可以由GRF内部转发解决。
  - 如果其与上次写入间隔一条指令，则无法在store指令的Mem级进行转发，因为此时上一条写入指令的WB阶段已执行结束，但是上次写入发生在本次读取之后，存在冒险，需要转发。可以考虑将转发行为提前，在store指令的EX阶段，对待写入DM的值的寄存器在WB级流水检索是否存在可用转发源，若存在则将其转发过来，替代原本从GRF读出的旧值，若不存在则不作处理。
    - 注意这里接入ALUIn1的数据时可以选择接入转发前直接读出的数据ALUIn1或转发后的数据ALUIn1_bypass，但是我们后面在Mem入口处已经设定了转发，这里如果接入ALUIn1_bypass的话，一是难以判断是否转发，二是与后面重复，因此我们选择接入直接读出的数据ALUIn1。
    - 考虑将其与ALUIn1的转发通道合并，将ALUIn1的转发MUX改为不管Tuse的值是否为0都扫描后面有无可用转发源
  - 如果其与上次写入为相邻两条指令，则在store的Mem阶段上一条指令已经走到WB，必然已经得到结果，可以从WB向Mem转发，因此DM入口处再检测一次是否能从WB转发即可。
  - 需要注意的是，即使存在连续几条指令都对同一个寄存器写入，最后一条指令需要将该寄存器的值存回DM的情况，上述转发方案仍然成立，因为每次store指令的转发都是用最新产生的数据覆写store的数据流。

4、加指令工程化步骤

（1）修改头文件，加入新指令的独热码；

（2）修改InstrDecoder，加入新指令译码；

（3）修改AT_Cal，加入新指令的AT计算环节；

（4）

- 若加入的是跳转，则修改ID中的Jump信号产生模块
- 若加入指令在ID级产生值，则修改ID中产生结果的模块
- 若加入的是分支，修改ID中Branch信号产生模块
- 若加入的是ALU相关计算指令，则修改ALU信号
- 若加入的指令涉及立即数运算，在ALU信号模块中修改ALU的数据来源以加入立即数

5、乘除指令冲突处理

- 当前一条指令为乘除类指令时
    - 若在Busy或Start信号有效时在ID级译码发现有与乘除模块相关指令出现，则暂停执行，直到Busy信号解除。
    - 其它情况不会出现冲突，因为没有其它指令会直接读取/写入HI或LO。
- 将乘除模块的D1和D2分别接在ALU的入口上，则它们的输入可以看作ALu的输入，共享与ALU的转发。
- 将乘除模块的输出接在ALU输出选择上，当mflo和mfhi指令出现时，ALU输出选择相应来源即可。
- mult、div、multu、divu、mtlo、mthi指令Tnew统一设置为0，Tuse设为相应利用时间；mflo、mfhi指令不需要Tuse，Tnew设置为从EX级产生结果。

6、异常信号产生

每一级产生各自的异常信号并接收从前级流水的异常信号。

正常情况下用本级异常信号覆盖前级异常信号。

特殊地，在Mem级接收到溢出信号并发现对应指令是load或store类时，将异常信号更改为存取错误。

最后在Mem级统一将异常信号和类型提交CP0。

二、CP0设计

（一）接口设计

CP0的模块接口如下所示。

信号名	方向	描述
clk	I	时钟信号
reset	I	同步复位信号
InstrType[59:0]	I	Mem级指令
WBInstrType[59:0]	I	WB级指令
PCAddr[31:0]	I	Mem级PC
DataIn[31:0]	I	CP0写入数据
CP0Addr[31:0]	I	CP0寄存器读写地址
Err	I	内部错误信号
ErrStat[4:0]	I	内部错误类型
HWInt[7:2]	I	外部中断信号
ErrSignal	O	全局中断信号
DataOut[31:0]	O	CP0读出数据

（二）关键功能实现

CP0寄存器：在CP0内部实现，在CPU需要读取时通过CP0Addr传入地址后，从DataOut传出数据交给CPU；需要写入时，InstrType显示当前指令为mtc0，CP0通过检查CP0Addr的地址决定将DataIn的数据写入相应寄存器。

中断处理：Err为CPU内部异常信号，HWInt为外部中断信号。当它们存在有效位，且全局中断IE开启，EXL关闭时启动中断，通过ErrSignal信号向CPU传递中断发生信号，并在内部向EPC、Cause、SR寄存器写入相应值；遇到eret时，SR相应将EXL置零。

三、桥与IO设计

（一）接口设计

Bridge的模块接口如下所示。

信号名	方向	描述
PrAddr[31:2]	I	CPU的IO地址
PrWD[31:0]	I	CPU向IO写数据
PrRD[31:0]	O	CPU从IO读数据
PrWE	I	CPU向IO写数据使能
Addr_0[31:2]	O	定时器0操作地址
WE_0	O	定时器0写使能
Din_0[31:0]	O	定时器0写入数据
DOut_0[31:0]	I	定时器0读取数据
IRQ_0	I	定时器0中断信号
Addr_1[31:2]	O	定时器1操作地址
WE_1	O	定时器1写使能
Din_1[31:0]	O	定时器1写入数据
DOut_1[31:0]	I	定时器1读取数据
IRQ_1	I	定时器1中断信号
Interrupt	I	外部中断信号

（二）关键功能实现

Bridge是纯组合逻辑电路。其检测CPU的操作地址，并根据操作地址将相应的写入信号、写入数据、读取数据分配到相应的接口。此外，Bridge还将外部中断信号集合起来形成HWInt[7:2]的总体外部中断信号。

四、测试方案

测试1：内部异常

主程序：

```
.text
lui $4, 0x7fff
ori $5, $0, 0xffff
ori $6, $0, 5
add $7, $4, $5
ori $8, $0, 0x3024
#lw $7, 0x1234($0) # Overflow Here
#jr $8
bgez $8, bdest
add $9, $7, $6
lui $7, 0x1234
ori $7, $7, 0x5678
bdest:
lui $7, 0x2234
```

```
ori $7, $7, 0x5678
lui $7, 0x3234
ori $7, $7, 0x5678
lui $7, 0x4234
ori $7, $7, 0x5678
```

中断处理程序（跳下一条）：

```
.ktext 0x00004180
#addi $20, $0, 14
mfc0 $1, $14
ori $2, $0, 0x4ffc
subu $2, $1, $2
bgez $2, start1
nop
addiu $1, $1, 4
end1:
mtc0 $1, $14
eret
start1:
ori $1, $0, 0x301c
j end1
nop
```

测试2：外部中断

主程序：

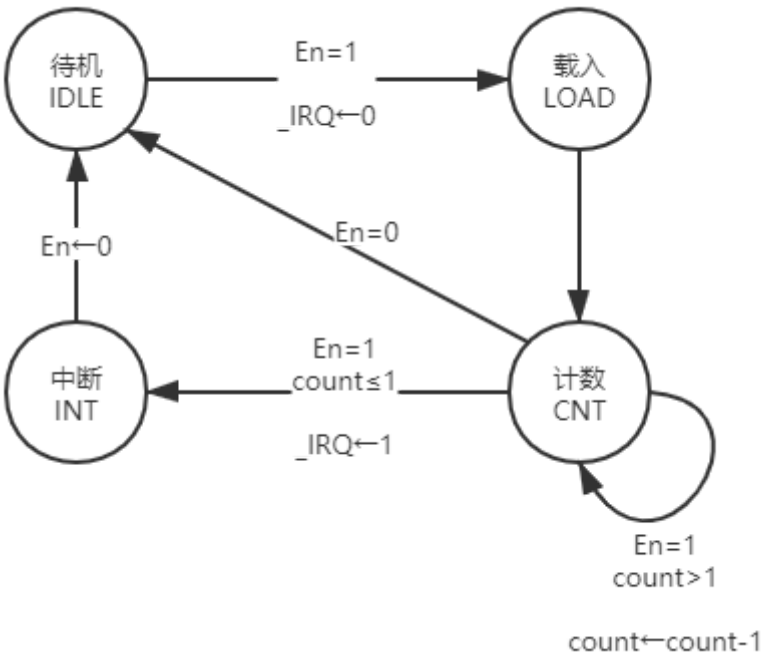
```
.text
ori $11, $0, 1 #IM
sll $11, $11, 3
ori $12, $0, 0 #Mode
sll $12, $12, 1
ori $13, $0, 1 #Enable
addu $14, $11, $12
addu $14, $14, $13 #CTRL

ori $15, $0, 100 #PRESET
ori $10, $0, 0x7F00 #Start address
ori $16, $0, 150
sw $15, 4($10)
sw $16, 0x7f14($0)
sw $14, 0($10)
sw $14, 0x7f10($0)
LoopStart:
addiu $5, $5, 1
jal LoopStart
nop
nop
```

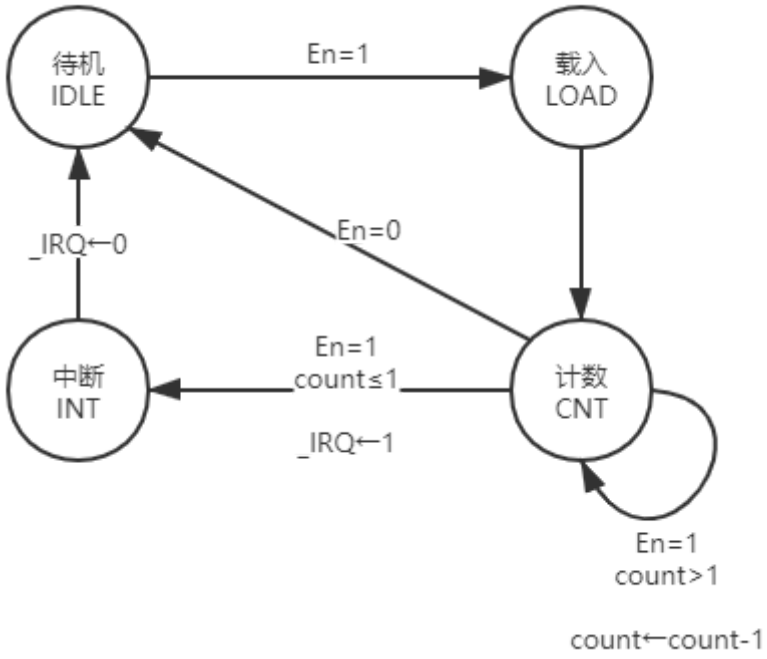
中断处理程序：

```
.ktext 0x00004180
lw $1, 0x7f00($0)
srl $1, $1, 1
sll $1, $1, 1
addiu $1, $1, 1
sw $1, 0x7f00($0)
addiu $6, $6, 1
eret
```

五、思考题

- 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”?
  - 接口指的是计算机不同功能层之间的通信标准。硬件/软件接口即指的是硬件层与软件层之间的通信标准。至今为止我们都在硬件层面上搭建CPU以满足能让MIPS程序在我们的CPU上运行的目的，MIPS标准就是我们的硬件/软件接口，沟通了我们的CPU和MIPS程序。
- 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。
  - 现代计算机中，DM应该是我们常见的内存条之类设备。但是考虑到提高处理速度，CPU内配置了多级缓存，也可以看做DM的一部分。
- BE 部件对所有的外设都是必要的吗?
  - 不是。对于一些必须整字读写的外设就不是必要的，此类设备只能采用lw和sw读写，其它读写方式应被认为是错误的。
- 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
  - 中断模式0和1都将从初值寄存器中载入初值然后计数，不同之处在于：
  - 模式0只计数一次，计数倒数至0时将激活中断信号，进入待机（IDLE）模式，并将计数使能关闭，直至下一次用户主动开启计数使能再重新从初值寄存器载入并开启一次新的计数周期；
  - 模式1计数倒数至0时激活中断信号，中断信号保持一周期后自动从初值寄存器载入初值，开启新的计数周期并清除中断信号，直至计数使能被用户主动关闭为止。
  - 状态转移图分别如下。
  - 

中断模式0的状态转移图



中断模式1的状态转移图

- 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：
  - 定时器在主程序中被初始化为模式0；
  - 定时器倒计数至0产生中断；
  - 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。
  - 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。
    - 见测试程序中的外部中断部分。
- 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？
  - 鼠标和键盘输入时，向CPU发送一个外部中断信号。CPU在捕获到外部中断信号后将对应的操作存入队列并结束中断。之后再回到操作系统层面逐个处理。