计算机组成原理实验报告

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，支持的指令集为：

MIPS-C4={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、SUBU、MULT、MULTU、DIV、DIVU、SLL、SRL、SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO、MFC0、MTC0、ERET}。

CPU 支持中断和异常。

（二）关键模块定义

1.CP0

CP0 根据当前的异常信号和外部中断信号判断是否产生中断。端口定义如表 1 所示。

表 1 CP0 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
A[4:0]	I	读/写 CP0 的寄存器编号
DI[31:0]	I	CP0 的写入数据
PCI[31:0]	I	中断/异常时的 PC 值
ECI[6:2]	I	中断/异常的类型
HWInt[7:2]	I	6 个设备中断信号
WE	I	CP0 寄存器写使能

ERET	I	当前是否为 ERET 指令
BDI	I	当前是否为延迟槽指令
IntReq	O	中断信号
PCO[31:0]	O	输出的 EPC
DO[31:0]	O	CP0 读出的数据
Respond	O	是否响应外部中断

2.Bridge

Bridge 实现 CPU 与外部设备的沟通。在 P7 主要包括两个计时器和 DM。端口定义如表 2 所示。

表 2 Bridge 端口描述

端口名称	方向	功能描述
IntReq	I	CP0 的中断信号
Respond	I	是否响应外部中断
interrupt	I	从 tb 传来的中断信号
A[31:0]	I	读/写的地址
DI[31:0]	I	写的数据
SOp[2:0]	I	写入的指令类型
LOp[2:0]	I	读出的指令类型
DM_Byteen[3:0]	O	DM 的字节使能信号
DM_A[31:0]	O	DM 的读写地址
TC0_WE	O	TC0 写入使能
TC1_WE	O	TC1 写入使能
DM_DI[31:0]	O	DM 最终写入的值
DM_DO_[31:0]	I	DM 读出的原始值
TC0_DO[31:0]	I	TC0 读出的值
TC1_DO[31:0]	I	TC1 读出的值
DO[31:0]	O	从 Bridge 整体读出的值
IRQ0	I	TC0 的中断信号

IRQ1	I	TC1 的中断信号
HWInt[7:2]	O	外部中断信号
EC[6:2]	O	异常类型

3.CU

CU 根据当前指令判断当前 CPU 需要进行什么操作，根据不同指令的需求通过输出信号控制 CPU 各部分的运行。同时在 CU 里得到读取、写入的寄存器地址，便于后面处理。端口定义如表 3 所示。

表 3 CU 端口描述

端口名称	方向	功能描述
S[31:0]	I	输入的指令
Stage[1:0]	I	当前在哪个阶段
NPCOp[2:0]	O	控制 NPC 的操作
GRF_D_Op[1:0]	O	选择 GRF 的写入数据
EXTSign	O	EXT 的扩展方式
ALUOp[2:0]	O	ALU 的计算类型
ALU_B_Op	O	ALU 的第二个计算数的来源
DMWrite	O	DM 的写入信号
A1[4:0]	O	需要读取的第一个寄存器
A2[4:0]	O	需要读取的第二个寄存器
A3[4:0]	O	需要写入的寄存器
Tnew[1:0]	O	见（三）3.
Tuse1[1:0]	O	A1 的 Tuse，见（三）3.
Tuse2[1:0]	O	A2 的 Tuse，见（三）3.

4. ALU

ALU 提供除了乘除外 32 位所有运算功能。其端口设置如表 4 所示。

表 4 ALU 端口描述

端口名称	方向	功能描述
Op[3:0]	I	代表需要进行的运算操作

A[31:0]	I	进行运算的第一个数
B[31:0]	I	进行运算的第二个数
C[31:0]	O	运算结果
Overflow	O	是否溢出

5.MDU

MDU 负责实现乘除操作，内置了 HI、LO 两个寄存器。其端口设置如表 5 所示。

表 5 MDU 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
Start	I	乘除运算的开始信号
Op[3:0]	I	代表需要进行的运算或存取操作
A[31:0]	I	进行运算的第一个数
B[31:0]	I	进行运算的第二个数
C[31:0]	O	运算结果
Busy	O	当前是否正在进行运算的信号

6. GRF

GRF 负责读写 32 个寄存器，编号为 0~31，其中 0 号寄存器的值始终保持为 0，其它寄存器的初始值为 0。GRF 可以随时读出 2 个不同寄存器的内容，并在信号的控制下在时钟上升沿将指定内容写入指定寄存器，具有同步复位的功能。其端口设置如表 6 所示。这里把写使能信号优化了，在不写入的情况下保证 A3=0。

表 6 GRF 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
A1[4:0]	I	指定 32 个寄存器中的一个，将其中存储的数据读出到 DO1

A2[4:0]	I	指定 32 个寄存器中的一个，将其中存储的数据读出到 DO2
A3[4:0]	I	指定 32 个寄存器中的一个，作为写入的目标寄存器
DI[31:0]	I	需要写入 A3 指定的寄存器的数据
DO1[31:0]	O	32 位数据输出信号，表示 A1 指定的寄存器中的数据
DO2[31:0]	O	32 位数据输出信号，表示 A2 指定的寄存器中的数据

7. PC

PC 为寄存器，存储当前指令地址，具有同步复位功能，起始地址为 0x00003000。其端口设置如表 7 所示。

表 7 PC 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
Stall	I	阻塞信号
IntReq	I	中断信号
ERET	I	ERET 信号
PCI[31:0]	I	下一步的指令地址
PCO[31:0]	O	当前输出的指令地址
Err	O	是否有取指错误

8.NPC

NPC 根据当前指令的内容计算下一条指令的地址。其端口设置如表 8 所示。

表 8 NPC 端口描述

端口名称	方向	功能描述
NPCOp[2:0]	I	控制信号
eq	I	ALU 计算结果是否为 0。用于判断 beq。
PC[31:0]	I	当前的指令地址（F 级）
Imm[25:0]	I	（可能）需要跳转的立即数
Ra[31:0]	I	（可能）需要跳转的寄存器值

NPC[31:0]	O	下一步指令地址

9.EXT

分别实现立即数的 0 扩展以及符号扩展。随后输出作为 ALU 的运算数。其端口设置如表 9 所示。

表 9 EXT 端口描述

端口名称	方向	功能描述
EXTSign	I	扩展方式控制信号
DI[15:0]	I	需要进行扩展的数
DO[31:0]	O	扩展后的数

（三）重要机制实现方法

1. 跳转

CPU 支持 beq、jal、jr、j 跳转指令。

本质上就是在 NPC 模块中加入一个控制信号 NPCOp，然后判断下一步的地址。

对于 beq，需要在 ALU 中判断两个寄存器的数值是否相等，若相等则将立即数进行符号扩展、左移的处理，作为偏移量加到 PC 值上。

对于 jal、j 和 jr，就可以直接赋值成对应的地址。

2. 流水线延迟槽

当前指令为 beq、j、jal 或 jr 时，会产生跳转，而此时 F 级已经读出新的指令，这条指令也会被执行，到下一个周期才会跳转到指定地址。这就是延迟槽。由于延迟槽，jal 需要写入 PC+8 至$ra。

为了增大效率和保持统一，把 beq 的比较前置到 D 级，在 GRF 读出数之后就进行比较。

在 NPC 计算时，我使用 F 级的 PC，D 级的控制信号进行计算。

3. 转发和暂停

首先明确转发数据的供给者和接收者。

供给者有：D/E 级寄存器（PC8 数据），E/M 级寄存器（PC8 或 ALU 计算

结果），M/W 级寄存器（PC8 或 ALU 计算结果或 DM 读出数据）。

接收者有：D 级 GRF 的输出（CMP 的输入）、E 级 ALU 的输入，M 级 DM 的输入。

在整体架构中明确两部分，第一部分是需要用来计算、存储的数据，是接收者；第二部分是已经得到，将要写入寄存器的数据，是供给者。

先假设没有暂停的情况。我们采用暴力转发的方案。即：每个接收者会接收到所有在它之前得到的数据。判断时只要需要使用的数据来源地址和某个将要写入的地址相同即可转发。多个满足条件的取最新的数据。注意 0 号寄存器不被转发，也不接收。具体可以见思考题（三）。

如果需要暂停，也就是某个地址的数据还没有被得到，但是马上就要在下一轮被用到。这里用 AT 法判断是否暂停。每个指令对应着两个读取的寄存器的 Tuse。Tuse 指从 D 级开始，再过多少周期后，数据一定要被转发到当前阶段，才能进行下去。例如，addu 的两个 Tuse 均为 1，sw 的一个 Tuse 为 1，一个 Tuse 为 2。

同时还有 Tnew。Tnew 表示在当前阶段，再过多少个周期后，才能产生数据并传到下一级寄存器中。例如，lw 在 D 级的 Tnew 为 3，E 级为 2。注意 Tuse 和 Tnew 的意义都是建立在确定的寄存器地址上的，比较的时候也要比较相同地址的情况。

我们先把所有指令拦在在 D 级，然后进行比较。如果两个需要的寄存器都满足 Tuse>=Tnew，那么就说明可以通过转发解决。否则，就必须在这里停顿，让前面的指令再流一个周期。这就是暂停机制。暂停时，PC 和 F/D 级寄存器不变，D/E 级寄存器清空，后面的照样流下去。

暂停机制确保了暴力转发的正确性。因为暴力转发的时候没有判断一个数据是否已经产生。在暂停机制下，如果数据没有产生就会被迫暂停，所以暴力转发是对的。

4. 中断处理程序

主要依靠 CP0 部件来实现。我把 CP0 放在 M 级进行处理。

首先考虑异常信号。在流水线中，F、E、M 级都可能产生异常，选择最早的一次发生异常的 ExcCode。流水到 M 级时，便可以传入 CP0。而对于外部中

断信号，通过桥的整合，传到 CP0 内。

CP0 根据外部中断信号和异常信号产生中断的信号，经过判断进入异常处理程序。对于异常产生的中断请求，如果当前不在中断状态，那么可以响应。对于外设的中断请求，如果 SR 寄存器中没有屏蔽，并且当前不在中断状态，那么可以响应。

在响应的时候，CPU 立即跳转至异常处理地址 0x00004180，在 M 级及之后的指令不被执行，而 M 级之前的指令保证执行。进入异常处理程序时，将 EPC、BD、ExcCode 等信息存入 CP0 的寄存器内。如果在延迟槽指令，则存的 EPC 为前一条指令。

在异常处理结束时，会通过 eret 指令回到 EPC+4。退出的时候也需要清空没有执行的指令。

二、测试方案

（一）典型测试样例

1. 阻塞的时候产生外部中断

tb 中设置宏观 PC 在 0x3018 时中断。

```
.ktext 0x4180
_entry:
      mfc0   $k0, $14
      mfc0   $k1, $13
      ori $k0, $0, 0x1000
      sw  $sp, -4($k0)

      addiu  $k0, $k0, -256
      move   $sp, $k0

      j   _save_context
      nop
```

```
_main_handler:
      mfc0   $k0, $13
      ori    $k1, $0, 0x007c
      and $k0, $k1, $k0
      beq    $0, $k0, _restore_context
      nop
      mfc0   $k0, $14
      addu   $k0, $k0, 4
      mtc0   $k0, $14
      j   _restore_context
      nop

_restore:
      eret

_save_context:
      sw     $1, 4($sp)
      sw     $2, 8($sp)
      sw     $3, 12($sp)
      sw     $4, 16($sp)
      sw     $5, 20($sp)
      sw     $6, 24($sp)
      sw     $7, 28($sp)
      sw     $8, 32($sp)
      sw     $9, 36($sp)
      sw     $10, 40($sp)
      sw     $11, 44($sp)
      sw     $12, 48($sp)
      sw     $13, 52($sp)
      sw     $14, 56($sp)
```

```
        sw      $15, 60($sp)
        sw      $16, 64($sp)
        sw      $17, 68($sp)
        sw      $18, 72($sp)
        sw      $19, 76($sp)
        sw      $20, 80($sp)
        sw      $21, 84($sp)
        sw      $22, 88($sp)
        sw      $23, 92($sp)
        sw      $24, 96($sp)
        sw      $25, 100($sp)
        sw      $26, 104($sp)
        sw      $27, 108($sp)
        sw      $28, 112($sp)
        sw      $29, 116($sp)
        sw      $30, 120($sp)
        sw      $31, 124($sp)
        mfhi    $k0
        sw  $k0, 128($sp)
        mflo    $k0
        sw  $k0, 132($sp)
        j   _main_handler
        nop



_restore_context:
        lw  $1, 4($sp)
        lw      $2, 8($sp)
        lw      $3, 12($sp)
```

```
lw     $4, 16($sp)
lw     $5, 20($sp)
lw     $6, 24($sp)
lw     $7, 28($sp)
lw     $8, 32($sp)
lw     $9, 36($sp)
lw     $10, 40($sp)
lw     $11, 44($sp)
lw     $12, 48($sp)
lw     $13, 52($sp)
lw     $14, 56($sp)
lw     $15, 60($sp)
lw     $16, 64($sp)
lw     $17, 68($sp)
lw     $18, 72($sp)
lw     $19, 76($sp)
lw     $20, 80($sp)
lw     $21, 84($sp)
lw     $22, 88($sp)
lw     $23, 92($sp)
lw     $24, 96($sp)
lw     $25, 100($sp)
lw     $26, 104($sp)
lw     $27, 108($sp)
lw     $28, 112($sp)
lw     $29, 116($sp)
lw     $30, 120($sp)
lw     $31, 124($sp)
lw  $k0, 128($sp)
mthi   $k0
```

```
        lw  $k0, 132($sp)
        mtlo   $k0
        j   _restore
        nop

    .text
    ori  $2, $0, 0x1001
    mtc0  $2, $12
    ori $28, $0, 0x0000
    ori $29, $0, 0x0000
    lui $8, 0x7fff
    lui $9, 0x7fff
    sw $8,8($zero)
    lw $8,8($zero)
    add $10, $8, $9
    or $10, $8, $9

    end:
    beq $0, $0, end
    nop
```

2. 阻塞的时候产生 ERET

tb 中设置宏观 PC 在 0x4198 时中断。

```
.ktext 0x4180
mfc0 $k0, $14
addiu $k0, $k0, 4
mtc0 $k0, $14
lui $k0, 12
lui $k1, 2131
div $k1, $k0
```

```
eret
mflo $k0
lui $20, 0x7654
lui $21, 0x8654
lui $22, 0x9654



.text
li $5, 0x0000ff11
mtc0  $5, $12
li $2, 0x7fffffff
li $3, 0x5ff
add $2, $3, $2
lui $16, 0xabcd
lui $17, 0x1234
```

3. 进入中断时清空还未进行的乘除槽

tb 中设置宏观 PC 在 0x300c 时中断。

```
.ktext 0x4180
……


.text
mtc0 $0,$12
ori $2,$2,0x1001
mtc0 $2,$12
mult $2,$2
```

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？)

答： 硬件接口是指同一计算机不同功能层之间的通信规则。比如外设和 CPU 之间通信需要硬件接口。

软件接口是指对协定进行定义的引用类型。其他类型实现接口，以保证它们支持某些操作。比如 API 等。

通过接口可以连接两个不同的设备、系统等，使得它们彼此间相互独立，却又可以传输数据。

（二）BE 部件对所有的外设都是必要的吗？

答：不是。比如 P7 实现的计时器部件，实际上只允许写入、读出整个字，半字和字节都是异常操作，因此 BE 对计时器不是必要的。

（三）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。

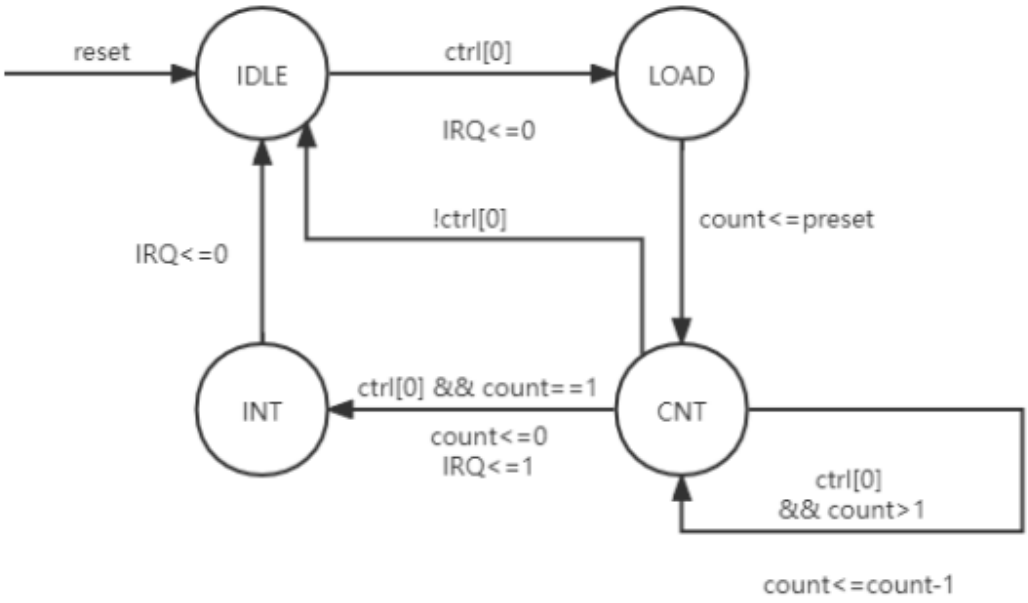
答：相同之处：两者在 IDLE、LOAD 和 CNT 状态的行为相同，且都在 count 到 0 的时候产生中断。

不同之处：两者中断的方式不同。

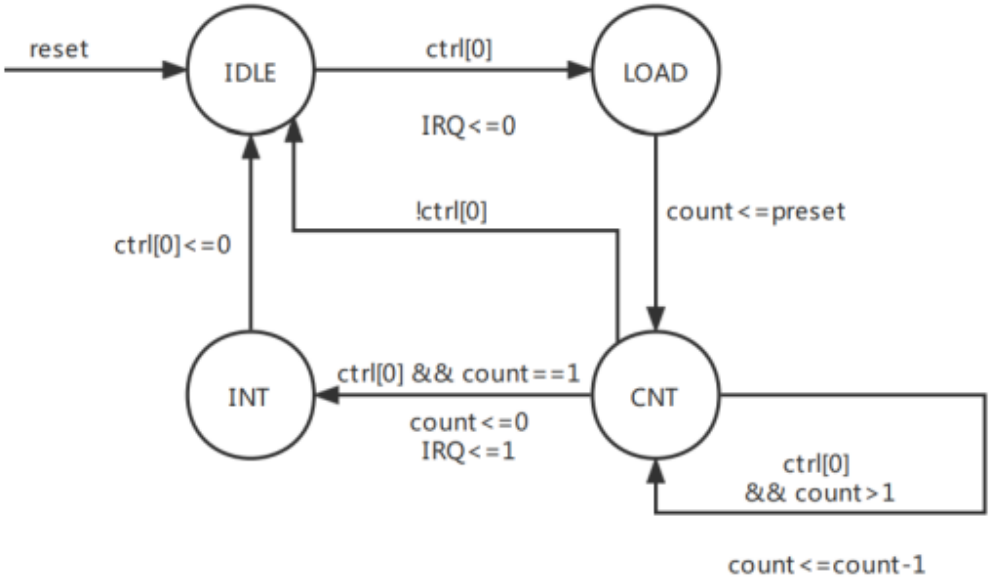
模式 0 计时结束后，一直保持中断，直到 reset 或状态被修改

模式 1 计时结束后，中断一个周期，然后回到初始状态，重新计数。

模式 0：用于产生持续中断信号，此时 ctrl[2:1] == 2'b00。



模式 1：用于产生周期性中断信号，此时 ctrl[2:1] != 2'b00。



（四）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。（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。）

答：

```
.ktext 0x4180
    li $t0,0x7f10
    li $t2,9
    sw $t2,0($t0)
    eret

.text
    li $t0,0x7f10
    li $t1,100
    sw $t1,4($t0)#preset

    li $t2,9
    sw $t2,0($t0)#ctrl

    li $t3,0xfa01
    mtc0 $t3,$12#SR

    loop:
    nop
    j loop
    nop
```

（五）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？

答：使用鼠标、键盘输入信号时，会发出外部中断信号，CPU 根据不同的中断信号执行对应的中断响应程序，在中断相应程序中进行相应的操作，把鼠标和键盘输入的信息写入寄存器，最终写入内存。