计算机组成原理实验报告

19373106 裴宝琦

一、CPU 设计方案综述

本 CPU 为 Verilog 实现的流水线 MIPS – CPU。

第一阶段：先构造出来不考虑转发暂停的流水线主控制器。

先后问题 发现 jal 和转发分开再处理会极度麻烦 所以直接用 AT 法。。

冒险表格：

	Tnew	Tusers	Tusert
add	1	1	1
sub	1	1	1
ori	1	1	3
lw	2	1	3
sw		1	2
beq		0	0
lui	1	1	3
j			
jal			
jr			
nop			

这里把 i 型指令都调到 3 了 防止误判暂停

最后的项目总结：基本的模块都是对应相应的功能 主要是根据 5 个转发位点冒险来建立对应的 MUX，我这里有的地方用了五级流水线的 12345，而不是 FDEMW，其实无区别，针对 jal/jalr 的值、R 类指令 ALUOUT 值的转发，解决了先后顺序的问题，然后根据 Tnew 和 Tuse 来触发暂停，暂停操作包括封锁 PC 的变化、IF/ID(1 级)的变化、清空 ID/EX(2 级)。

然后延迟槽的行为？有点迷惑

暂停最苛刻的情况 lw 接 beq 暂停两次

MULT、 MULTU、 DIV、 DIVU、MFHI、MFLO、MTHI、MTLO}

P6 接 P5 今天做简单的计算指令 ADD SUB SLL SLLV SRL SRA SRLV SRAV AND OR XOR NOR ADDI ADDIU ANDI XORI SLT SLTI SLTIU SLTU

1.增加 BED BEDop LED 扩展 DM

BEDop: 0:SW 1:SH 2:SB LB LH LBU

b) sw 指令：GPR[rt]写入对应的字。

地址[1:0]	BE[3:0]	用途
XX	1111	WD[31:24]写入 byte3 WD[23:16]写入 byte2 WD[15:8]写入 byte1 WD[7:0]写入 byte0

c) sh 指令：$GPR[rt]_{15:0}$写入对应的半字。

地址[1:0]	BE[3:0]	用途
0X	0011	WD[15:8]写入 byte1 WD[7:0]写入 byte0
1X	1100	WD[15:8]写入 byte3 WD[7:0]写入 byte2

d) sb 指令：$GPR[rt]_{7:0}$写入对应的字节。

地址[1:0]	BE[3:0]	用途
00	0001	WD[7:0]写入 byte0
01	0010	WD[7:0]写入 byte1
10	0100	WD[7:0]写入 byte2
11	1000	WD[7:0]写入 byte3

参考的接口定义如下：

信号名	方向	描述
A[1:0]	I	最低 2 位地址。
Din[31:0]	I	输入 32 位数据
Op[2:0]	I	数据扩展控制码。 000：无扩展 001： 无符号字节数据扩展 010：符号字节数据扩展 011：无符号半字数据扩展 100：符号半字数据扩展
DOut[31:0]	O	扩展后的 32 位数据

BE 模块和 LE 模块

增加判断溢出的操作（等待完善）

P7：

第一个部分：

CP0

3.3.1 状态寄存器（SR）

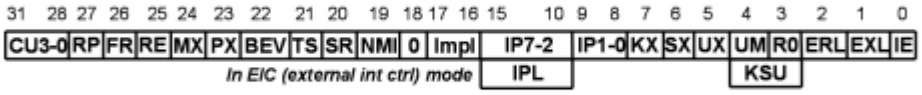


图 3.1: SR（状态）寄存器的各个域

必须实现的寄存器：SR,CAUSE,EPC，prid（12 13 14 15）

SR：IM7-2：由 CPU 核外的信号产生，1-0 为 cause 上可写的中断位

EXL：异常发生时置位 强行进入核心态禁止中断 决定 CPU

IE：中断使能位 EXL 禁止所有中断

CAUSE：

IP7-2：等待决定的中断 照抄输入信号 1-0 读写最近的值 告诉你现在发生了什么

其他位 0

EXCCODE:发生了哪种异常

BD:分支延迟

只要异常发生在延迟槽的指令，**Cause(BD)** 就会置位，**EPC** 就指向分支指令。如果想要分析异常受害指令，只要看看 **Cause(BD)** (如果 **Cause(BD)** == 1，那么该指令位于 **EPC**+4)就知道了。

EPC：写入字对齐

3.3.3 异常返回地址(EPC)寄存器

这只是一个保存异常返回点的寄存器。导致(或者遭受)异常的指令地址存入 **EPC**，除非 **Cause** 寄存器的 **BD** 位置位了，这种情况下 **EPC** 指向前一条（分支）指令。如果 CPU 是 64 位，么 **EPC** 也是 64 位。

PRID：

3.3.6 处理器 ID(PRId) 寄存器

图 3.3 显示了 **PRId** 寄存器的布局，这是一个标识 CPU 类型的只读寄存器。**CPU Id** 应当会各不相同——至少——当指令集或者控制寄存器定义发生变化时，**CPU Id** 的值要改变。“Revision”完全取决于制造厂家，只是用来帮助 CPU 厂家跟踪芯片版本，用做其它任何用途都是不可靠的。

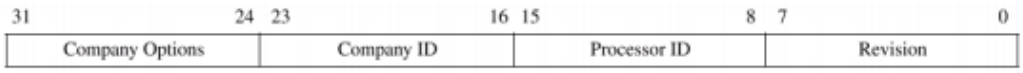
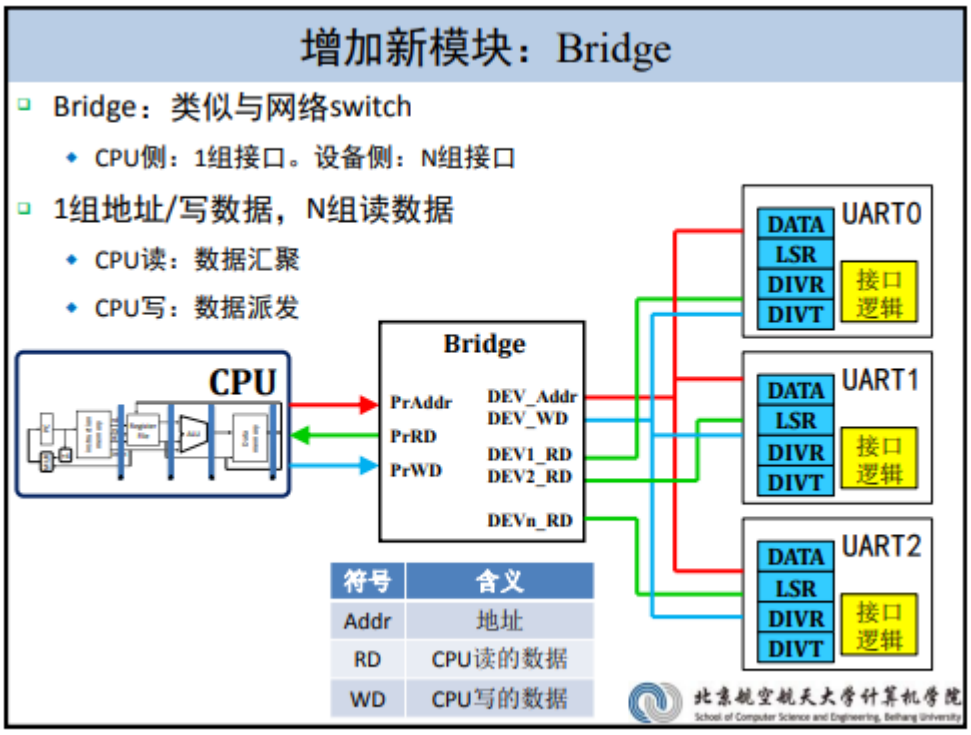


图 3.3: PRId 寄存器域

Company Id 域——可以从 MIPS 公司得到该域的取值——相对较新，历史上以往的 CPU 都将其设置为零。**Company Options** 只在你的 CPU 手册中有定义。

表 3.3 中列出了我们所知道的一些 **CPU Id** 的设置。

如果您想打印出这些值，习惯上写成“x.y”的形式，其中 x 和 y 分别为 **CPU ID** 和 **Revision** 的十进制值。尽量避免依赖这个寄存器的内容来确定某些参数（比如高速缓存大小，速度等等）或者确定某项特性的有无。有些特性在


增加新模块：Bridge
Bridge：类似与网络switch
CPU侧：1组接口。设备侧：N组接口
1组地址/写数据，N组读数据
CPU读：数据汇聚
CPU写：数据派发
CPU
Bridge
PrAddr
PrRD
PrWD
DEV_Addr
DEV_WD
DEV1_RD
DEV2_RD
DEVn_RD
UART0
UART1
UART2
DATA
LSR
DIVR
DIVT
接口逻辑
符号
含义
Addr
地址
RD
CPU读的数据
WD
CPU写的数据
北京航空航天大学计算机学院

建模 CP0

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？）

硬件接口：外设和 CPU 连接的接口，用来让两者产生联系

软件接口：用来接入软件来控制程序，让软件来控制设备

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。

   在现在的计算机中 DM 应该都是通过 CACHE 和 CPU 进行数据交换

2. BE 部件对所有的外设都是必要的吗？

   不是 比如这一次的 TIMER 寄存器，是读写整个寄存器，显然就不需要 BE。

3.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

见计时器设计文档

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 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。）

mips1.asm*

```
1  .ktext 0x4180
2  addi $a0,$0,0
3  ori $t0,$0,0x7f00
4  sw,$a0,0($t0)
5  ori $t2,9
6  sw $t2,0($t0) //设置控制寄存器
7  eret
8  .text
9  ori $t1,0x7f00
10 ori $a0,1
11 ori $t0,10
12 sw $t0,4($t1)
13 sw $a0,0($t1)
14 ori $a3,0xfc01
15 mtc0 $a3,12
16
```

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？

设备包括接口控制器和设备主体，CPU 并不是直接和外设联系，而是通过接口连接，当鼠标键盘输入时相当于中断，然后中断程序会把数据读到寄存器，从而获得信息。

处理过的一些 bug：

1.暂停应该屏蔽异常中断否则会进入不了 handler

2.MFC0 考虑寄存器为 0 的情况

3.DM 级不能乱写

4.宏观 PC 问题，延迟槽判断对应错了

5.没有考虑 MTC0 接 ERET 暂停。

6.lw 计时器