北京航空航天大學

BEIHANG UNIVERSITY

计算机组成原理实验报告

Verilog 支持 MIPS 微系统(含串口通信)

北京航空航天大学

计算机学院

姜翔舰

18373531

二〇一九年十一月

目录

第一章 模型顶层视图

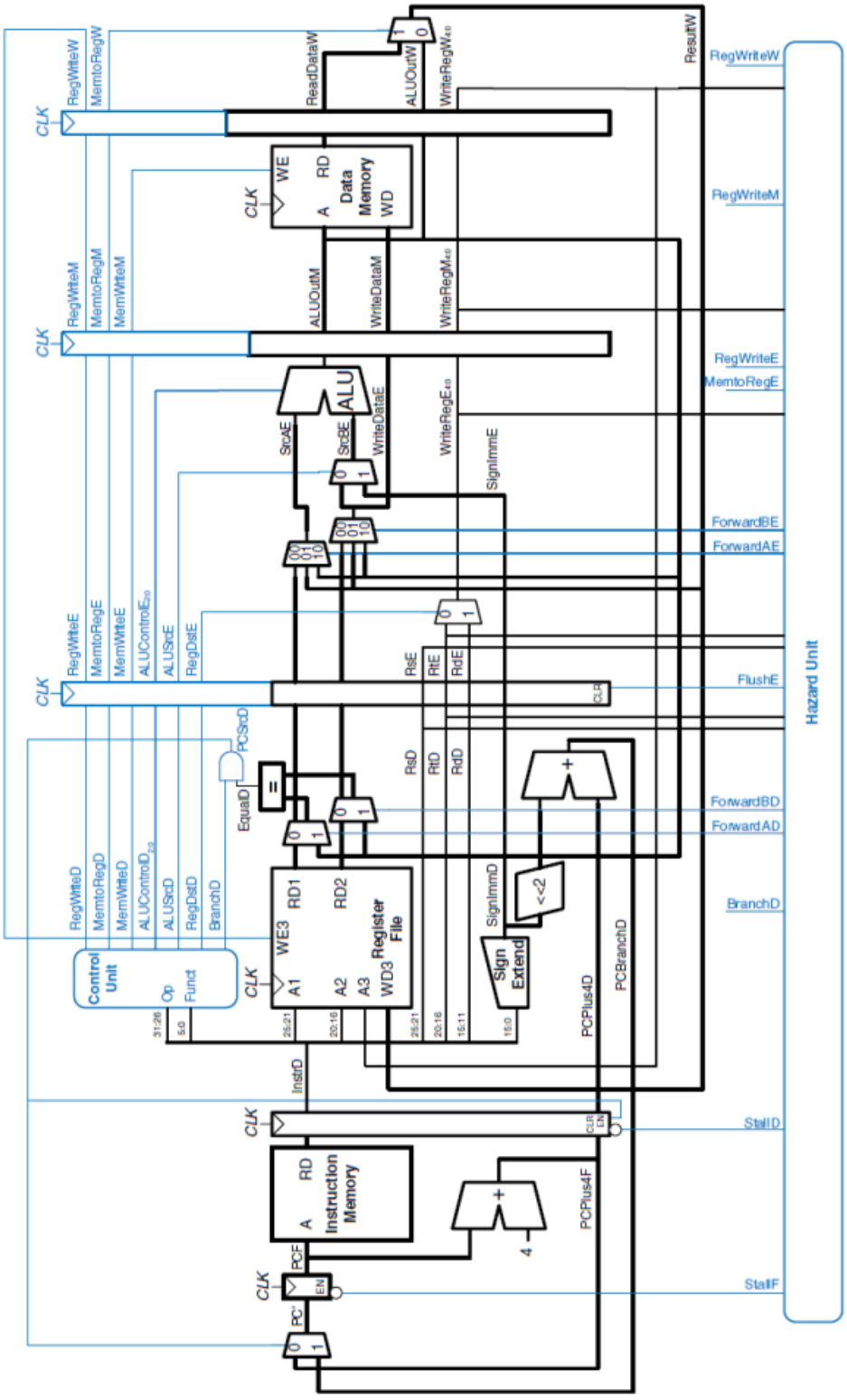

Figure 7.58 Pipelined processor with full hazard handling

MIPS (Microprocessor without interlocked piped stages)
流水线 CPU 数据通路基本模型

第二章 数据通路

1、实现方法

形式化建模的工程化方法

2、具体步骤

① 建立含有各个数据通路部件的分离模型.

② 根据指令集逐条确定各个部件的输入端口连接关系(接口关系)及功能要求.

③ 综合含有转发多选器的数据通路,整合部件功能.

3、注意事项

① 在确定接口关系并增加新部件时应遵循“高内聚低耦合”的思想,需要根据部件间的关系而不仅仅是功能分类进行连线.

eg.beq 指令的 NPC 地址计算并未由 ALU 完成,而是通过增加选择器将地址计算封装在 NPC 内,避免产生 ALU 和 PC 的接口关系使设计复杂化.

② 综合数据通路不仅需要确定端口的连接关系,同时整合某个部件需要实现的所有功能,并对功能进行排序编码.

③ 表格信号横向排序：**Op→Sel→We**

④ 模块端口声明顺序：

clk	1st
reset	1st
Op	2nd
Sel	3rd
We	4th

4、模块接口关系(表格见 datapath 文件夹)

5、模块功能列表及编号(表格 controller 文件夹)

6、模块规格

a) PC：

模块作用：输出程序当前执行指令在 IM 中的地址.

注意事项：

① PC 输出的是字节地址,IM 使用的是字地址,需要进行移位转化.

② PC 需要支持**同步**复位功能.

③ PC 输出相较单周期进行了 ADD4 和 NPC 的分离,因为 NPC 设计译码,被放置在了 ID 段.

PC 端口定义

Name / Interface	PC	
	Direction	Description
PC4	I	Normal PC+4
NPC	I	get branch or jump PC
PC	O	output PC

b)NPC
模块作用：计算出下一条指令的字节地址.
注意事项：NPC 是唯一驱动 PC 的组合逻辑部件,所以分支指令的地址计算需要封装进 NPC 中

NPC 端口定义

Name / Interface	NPC	
	Direction	Description
Ins	I	instruction
PC	I	get PC
equ	I	get sig whether two nums equal
NPC	O	output NPC

c)IM
模块作用：存储程序指令并输出当前执行指令.
注意事项：IM 是组合逻辑部件,当前指令在整个周期中都有效.

IM 端口定义

Name / Interface	IM	
	Direction	Description
PC	I	get PC
Ins	O	output instruction executing now

c)RF
模块作用：通用寄存器堆,进行指令中操作数的读写.
注意事项：

① RF 是时序逻辑部件,需要支持**同步**复位功能.
② RF 的内部转发可以被 W 向 D 的转发替代.

RF 端口定义

Name / Interface	RF	
	Direction	Description
Ins	I	instruction
RFWe	I	enable RF to be changed
RD1	O	output R[rs]
RD2	O	output R[rt]

d)ALU
模块作用：进行指令中的运算操作.
注意事项：A3 和 WD 端口的选择信号要严格按照生成的序号表搭建.

ALU 端口定义

Name / Interface	ALU	
	Direction	Description
A	I	operator1
B	I	operator2
ALU.C	O	result of calculation
Zero	O	whether two nums equal

f)DM
模块作用：内存读写
注意事项：DM 是时序逻辑部件,需要支持**同步**复位功能.

DM 端口定义

Name / Interface	DM	
	Direction	Description
Addr	I	addr of memory
WD	I	word to write
DMWe	I	enable DM to be changed
RD	O	data accessible in memory

g)EXT
模块作用：对立即数进行移位和扩展操作.
注意事项：对立即数的操作尽量封装到 EXT 中执行.

EXT 端口定义

Name / Interface	EXT	
	Direction	Description
Imm	I	Immediate num to be operated
Ext	I	Result after operation

第三章 主控制器设计

1、控制信号生成

方法：使用切片操作确定指令,并针对每条指令单独生成控制信号.

2、指令信号生成

a)具体步骤：

① 确定指令类型：R 型指令由 funct 生成,另外两种指令由 opcode 生成.

② 对指令进行切片提取操作,通过与运算生成指令信号.

b)注意事项：

① 生成指令时需要用**指令类型**对输出进行控制.

eg.I/J 指令的 imm 字段可能同时生成了一种 R 型指令,但是不能使改 R 型指令的输出信号有效,故需要将指令类型加入判断信号.反之,由于 R 的全零 opcode 字段不对应 I/J 指令,所以不用加入指令类型判断生成 I/J 指令.

② 控制器端口声明顺序：

Ins	1st
Op	1st
Sel	2nd
We	3rd

③ 生成操作信号时按照指令对应的信号表(含编号)进行生成.

3、操作信号生成

a)具体步骤：

① 根据指令集 RTL 描述生成指令信号

② 使用 if 语句生成指令对应的操作信号

③ 面向指令生成控制信号代码

b)注意事项：

① 操作信号直接使用常数生成,并且注释.

eg.ALUOp=5; //description of ALUOp5

② 每条指令需要对所有操作信号进行赋值,没有使用的统统当作 0 处理.

(不赋值会使指令信号值保持不变,造成操作信号混乱.)

③ 控制器中使用的操作序号和各个子模块中使用的操作序号必须保证相同.

4、具体实现图样

```
assign op=Ins[31:26];
assign func=Ins[5:0];
assign Rtype=(op==6'b000000);

//R
assign ADDU=Rtype&&(func==6'b100001);
assign SUBU=Rtype&&(func==6'b100011);
assign SLTU=Rtype&&(func==6'b101011);
assign JR=Rtype&&(func==6'b001000);
assign SLT=Rtype&&(func==6'b101010);
```

切片操作生成指令信号

```
if(ADDU)
    begin
        NPCOp=0;          //PC_4
        EXTOp=0;          //zero_ext
        ALUOp=0;          //add
        RF_A3_Sel=1;          //IM.D[15:11]
        RF_WD_Sel=0;          //ALU.C
        ALU_B_Sel=0;          //RF.RD2
        DM_WD_Sel=0;          //RF.RD2
        RFWe=1;      //1
        DMWe=0;
    end
```

面向指令的操作信号生成

第四章 CPU 测试

1、测试指令集

MIPS-C0 {addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}

2、测试程序设计须知

a) 测试全面

eg1.涉及数据比较时考虑++/+-/--三种情况

eg2.分支指令考虑朝前和朝后两种跳转方式

b) 指令条数不可超过 IM 内存限制

c) 不要使用指令集以外指令

d) 根据 Mars 进行比对

3、测试程序文档(见压缩包 test 文件夹)

4、测试流程

① 添加指令时逐条进行功能测试(不刻意考虑冒险,保证功能正确即可)

② 指令添加完成后根据 6 种转发,3 种暂停生成 9 种**特征模板**

③ 根据指令访存特征对模板进行替换即可

5、指令测试期望输出(见评测机综合的输出)

第五章 优化设计的思考

机器码 16 进制转化为 2 进制时**前 0 不能省略(保证位宽为 32 位)**

对 IM 和 DM 进行访问时 物理地址值(**字节**地址 eg.PC)需要**除以 4** 变为逻辑地址值(**字**地址)

更改指令条数后**不能**在 txt 文档里**直接删去**改行再对 CPU 进行测试(分支、跳转指令的地址可能发生变化)

指令的**非时序(eg.读取)**操作可以认为在上升沿前的低电平时段完成
eg.sb 可以取得内存相应字在模块外进行组合逻辑操作后等待上升沿写入内存

initial 块中均使用阻塞赋值

可重用函数的 automatic 放在 function **之后**
eg.function automatic [name];

MUX 采用 **parameter** 命名**操作**,避免冲突且方便修改.

指令全部用**大写**形式表示,防止和 verilog 关键字重复.

指令对应的每个操作信号后使用注释对常数进行说明

控制信号表格中,序号只能递增,不可以中途插入.

PC 的非阻塞赋值使得在 RFWe 失效**前**完成对寄存器堆的写入.

Ins 整体流水,解析在各流水级中分布完成.

内部转发可以被 **W** 向 **D** 的转发替代

暴力转发中 **load** 特征的内涵：指令在 **W** 级才能够确定要写入寄存器的数据.

转发和暂停的判断条件里一定要有各级的**写使能信号** 避免发生**不会被**写入数据的**误**转发或者暂停

stall_load 和 stall_beq 可以同时有效(同一时刻可以有**不只一个**暂停特征有效)

$signed()作用的本质：使当前数最高位被解析为符号位
1、$signed(PC)+4+$signed(Imm)*4;
当 Imm 的位宽和 PC 不同时 加$signed 可以使 imm 进行符号扩展,从而和 PC 相加求出正确结果

2、$signed(B)>>>shamt 实现算数右移
B>>>shamt 时,B 默认为无符号数,所以高位自动补 0.
3、$signed()对大小比较的影响
位宽不同时会进行**符号**扩展后再比较,位宽相同时则按有符号数直接比较.

负数表示 -32'd4

lwr/lwl/sw 等指令由于使用 rt 的阶段在 M 级,有足够时间供 load 指令取内存数据,所以即使与 load 指令发生冲突也不用暂停.

对于访存方式比较特殊的指令(eg.movz/lwpl)采用 both 属性,本质是保证了控制信号从 **D 级开始**就**不会**发生变化,所以不用更改冒险控制模块特判对该指令的暂停.

madd 是特殊类型的 R 指令(op 字段不为 0)

只有时序逻辑部件需要 timescale 确定时间粒度

wire/reg 和负数比较时,如果位宽相同,则与有没有$signed()无关

无符号乘法和有符号乘法的区别体现在**负负相乘**

对于可能不经过写入**直接读取**的寄存器来说,需要初始化.
eg.RF/HI/LO 需要初始化,但是流水寄存器不需要.

output reg 的本质是**当前模块**的一个**寄存器**,所以可以用在模块内作为判断条件(取值是上升沿**前**的值).

添加指令的步骤：
①写出 RTL 级操作
②在 MARS 中确认指令行为(特别注意链接寄存器是否需要满足跳转条件)
③建立 datapath(**新增端口**在表格中高亮)
④建立控制信号表
⑤更改 datapath(修改完模块后立即在顶层进行相应修改 **新增一个端口就声明一个信号**)
⑥生成控制信号
⑦本地测试(全面)

IM 只限制外部中断,并不限制异常的处理.

CP0 可以想象放在 M 级

判断溢出时 RST 和 RST_ext 的比较需要**带符号**

$signed(A+B)和$signed(A)+$signed(B)
前者会计算出 A+B 并**截断后**再进行符号位扩展
后者会对结果直接进行符号扩展后赋给左值

CP0 中除 PRId 外初始化为全 0 即可

安全的 BD 位判断方式：D 级判断

DM_RD 和 CP0_RD **不合并**因为 CP0 并未在内存编址 但是 DM_RD 和 Pr_RD 可以合并

发生异常清空**所有**流水寄存器,包括 W 级(M 级异常指令也不能执行完)

异常/中断发生时,M 级指令不能对内存写入

Cause 是只读存储器,不能由程序员更改,只能由硬件自身信息(eg.异常)更改.

Cause 寄存器的外部设备编号：Timer0 输出的中断请求信号接入 **HWInt[2]** (最低中断位),Timer1 输出的中断请求信号接入 **HWInt[3]**,来自 MIPS 微系统外部的中断请求信号接入 **HWInt[4]**.

$15 是指编号为 15 的寄存器(不看$15 内的值)
GPR[$15]是寄存器的值

共阴级数码管是 0 信号点亮

异常中实现的操作采用先**统一跳转** 每个标签后使用 eret 的方法
更新值的操作放在外部循环中 具体运算放在异常里

第六章 暴力转发&工程化方法的思考

一、本质思想

二者的本质思想相同:

在前面指令的结果来得及被利用时,采用转发.

没有办法在新值准备好前利用时,采用暂停.

二、实现方法的区别

1、暴力转发

① 转发：不考虑当前指令的寄存器需求,只要机器码中出现数据冒险,则进行转发.因为暂停机制的存在,当指令真正使用寄存器值时,一定是新值；如果不使用,则新值也不会产生影响.

② 暂停：立足对所有指令访存方式的分析,提取出分立的三个特点:store/both/single 即可通过三者的组合实现任意的暂停机制.(在 Excel 表中通过观察得到该两行一列即可覆盖所有的情况)

2、工程化方法

立足已知指令集,将指令按照访存方式(Tnew&Tuse)分为不同的类别,然后具体分析转发和暂停.

① 转发：当指令需要才转发,**不需要则不转发**,但是可能存在隐性的需要没有得到满足(eg.sw 在 M 级使用 rt 如果仅仅在 M 级考虑对 rt 的转发,可能此时并没有发生冲突 (对 rt 的改变在 sw 使用前已经悄悄完成了) 所以 rt 并没有被更新.因此 rt 的转发需要在之前每一级都被考虑到).

② 暂停：根据每个指令类别具体行为比较时间后决定是否暂停.

三、暴力转发的加速思想

暴力转发和工程化方法相比,劣势就在于如果对指令的行为特征分析不全面,会造成无谓的暂停.(eg.lw 和 addi 的 rt 发生冲突,如果只根据 load 进行暂停判断,则会暂停,但其实不需要)

因此,暴力转发在三种指令特征中,均有加速优化点.

1、load

① 判断 rt 是否会被 ALU 使用(ALU_B_Sel==0)

Q: 是否存在指令不使用 alu 但是 ALU_B_Sel 的控制信号默认选择了 rt 而造成的多余 stall_load 的情况?

A:若 ALU_B_Sel 是被默认为 0,即不需要使用 ALU 的情况.则只可能是跳转指令或者分支指令,而当这些指令由于 load 在 E 级而暂停时,并不是因为 rt.所以,不存在 alu 不使用 rt 但是由于 ALU_B_Sel=0 造成的意外的 stall_load 有效

② 判断 load 写入的寄存器是否是$0(A3_E==0)

2、both

① 判断写入寄存器是否是$0(A3_E/A3_M/A3_W==0)

3、single

① 判断写入寄存器是否是$0(A3_E/A3_M/A3_W==0)

第七章 关于流水线 CPU 结构的思考

Q1:为什么单独设置了 ADD4 模块进行 PC 的普通更新,而不是为了维持结构的整体性从而始终让且仅让 NPC 驱动 PC?

A1:五级流水线将 PC 和 NPC 分隔在了不同的区域,由于中间的 D 流水级寄存器的存在,导致两者的值传递存在一个周期的延迟,即 NPC 得到的 PC 是上一个时钟周期内 PC 的值.所以 PC+4 不能由 NPC 完成,需要单独设置 ADD4 来完成普通更新.同时,这个时钟延迟导致了 PC(模块)更新为 D 级流水寄存器中的 PC(数值)对应的 NPC 输出的新地址的时间延缓了一个周期,这也是为什么需要为分支和跳转指令设置延迟槽的原因.而且,因为 NPC 和指令有关,所以不放在 IF 而是 ID 中.

Q2：延迟槽对 jal 和 beq 的具体影响.

A2：

① jal

D 中指令是 jal 时,NPC 计算出需要跳转到的 PC 值和需要被回跳的 PC 值.当下一个时钟上升沿来临时,jal 进入 E 级,但是进入 D 级的并不是需要被跳转到的 PC 处的指令,因为在刚刚到来的上升沿 PC 才完成更新,所以进入 D 级的是原本紧跟着 jal 的指令,也即是延迟槽指令.所以计算需要被回跳的 PC 值时,需要将 jal 对应的 PC+8,而不是+4.

② beq

D 中指令是 beq 时,对延迟槽指令的分析与 jal 相同.但是计算需要被跳转到的 PC 值时,并不需要使用 PC+8+offset.因为编译优化是指从高级语言到汇编语言时,对指令进行顺序上的调整,也即对延迟槽进行指令填充.所以对于已经填充好的汇编程序生成的机器码来说,CPU 在执行 beq 时对应的 offset 在 MARS 中已经将 offset 考虑在内,因此 beq 的 PC 计算公式并没有被延迟槽影响.

Q3：D 级单独设置 CMP 的原因以及和 ALU 中比较功能的区别.

A3：流水线 CPU 的控制冒险来源于分支语句执行方向的不确定性,如果让 beq 在流水线中流水过深,当需要发生跳转时则会造成流水线的强制排空,对性能产生较大影响.所以将 beq 的比较环节提前到 ID 部分可以是排空的损失尽量减小,如果配合延迟槽技术则可以实现不考虑暂停的全速流水线.因此设置 CMP 是为了服务于分支指令的 NPC,而 ALU 中的比较功能则是满足其他指令的操作(eg.slt).因此二者只是功能相似,但是驱动的部件和意义完全不相同.

第八章 工程环境注意事项

先运行 100ns 然后再 10ns 递进实现**指令调试**

新增一个其他指令都没有的控制信号时可以在 always 开头**单独**对该信号**置零**

由于 verilog 信号的不变性,nop 需要**单独**定义指令信号.

新增端口使用**下划线**命名法

检查 cpu 前先查看 **IM** 是否更新为 code.txt 中的内容

计算式中既含有符号数又含无符号数则全按**无符号数**处理,因此 beq 的指令计算中对每个寄存器使用$signed().

新建 verilog 项目后 add source 即可添加所有.v 文件,但不会改变.v 文件的路径.

切片区域值含变量时,若位宽为变量则采用**循环**进行位赋值,否则使用 prompt 格式进行切片.
eg.in[sel*4+3 : sel*4] does not work.
Prompt: in[(sel*4+3)-:4]

aligned adj.字对齐(字节地址是 4 的倍数)

返回结果为空时,可能是文件大小超过了 4MB.

暂停的三个操作：
① 冻结 **PC** 值
让 D 流水级指令对应的下一条指令保持不变.
② 冻结 **D** 流水级
③ 清零 **E** 流水级
同步清零,相当于不让 Ins_D 进入 E 并且插入了空指令.

转发的值**只能**来自寄存器,否则会发生振荡.

自定义模块生成的实例必须要命名,且不同文件中的实例名和端口名都可以相同.

不同.v 文件中的**宏**定义在**没有**互相 include 时不会有影响.

考虑延迟槽时,如果 jal 跳转到的指令和延迟槽指令是同一条,则延迟槽指令会被执行两次.

16KB 的 IM 超过了 MARS 的限制(MARS 最多 1024*4B=4KB)

MARS 中$28/$29 寄存器的初始值并不是 0,所以生成的测试程序需要首先将这两个寄存器置零.

$signed()在 always 和三目运算符中特性不同,统一使用 always 语句.

while 循环不能用 break 跳出,将所有跳出条件直接作为循环判断条件.

MARS 中不能处理延迟槽异常

第九章 转发暂停机制的分析

一、暂停机制(Stall)

1、暂停的意义

暂停是最为直白的解决数据冒险的方式.暂停即是在 Ins_D 在 ID 级完成译码后立即判断与其他流水级的指令是否会发生冲突,如果会,则暂停指令的更新,等待冲突消失后流水线再读取新指令.

2、暂停的判定

暂停的判断需要采用 Tuse/Tnew 模型.其中 Tuse 表示当前在 **ID** 级(只需要考虑 ID 级,因为每一条指令刚进入流水线时会发生的冲突就已经可以被判断出来了.)的指令对寄存器堆数据的使用还有几个时钟周期,Tnew 表示其余各级指令产生新寄存器值并存入流水寄存器所需的时钟周期.当产生数据冒险时,即对 Tuse 和 Tnew 进行判断,若 Tuse<Tnew 则说明没有任何办法使 ID 级指令在正常的时间获取需要的最新寄存器值,则采用暂停的方式实现 Tnew 的减小和 Tuse 的恒定不变.

3、暂停的实现

根据指令的访存特性采用暴力转发实现.

4、Tuse&Tnew 机制分析

①数据需求 Tuse：指令位于 **D 级**的时候,再经过多少个时钟周期就必须要使用相应的数据.

eg1. beq Tuse=0/addu Tuse=1

eg2. sw 在 EXE 级它需要 GPR[rs]的数据来计算地址,在 MEM 级需要 GPR[rt]来存入值,所以对于 rs 数据,它的 Tuse_rs=1,对于 rt 数据,它的 Tuse_rt=2.(一个指令可以有两个 Tuse 值)

②数据产出 Tnew：位于**某个**流水级的某个指令,它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里.

eg.对于 addu 指令,当它处于 EX 级,此时结果还没有存储到流水级寄存器里,所以此时它的 Tnew=1,而当它处于 MEM 或者 WB 级,此时结果已经写入了流水级寄存器,所以此时 Tnew=0.

特点 1：是一个**动态值**,每个指令处于流水线不同阶段有不同的 Tnew 值

特点 2：一个指令在一个时刻只会有一个 Tnew 值(一个指令只有一个结果)

二、转发机制(Forwarding)

1、转发的意义

转发是为了优化数据冒险的暂停解决而采取的机制.数据冒险的本质是指令通过寄存器堆交换数据.(且不包含流水线寄存器,因为流水线寄存器的有效数据最终都会写入寄存器堆)同时,对 DM 的读写操作均发生在 MEM 阶段,中间不会存在类似 RF 的 D 级读,W 级写的时间间隔供其他指令进行读取访问,所以产生数据冒险的只可能是 RD1 和 RD2 对应的 RS 和 RT 寄存器.

2、转发的判定

转发的理论基础和暂停相同,均是 Tnew/Tuse 模型.在暂停中,我们只需要分析 Ins_D 和后面各级指令的类型关系即可,因为暂停与否取决于 Ins_D 的 Tuse 和其他各级的 Tnew 的关系.但是在转发机制中,在 D/E/M 级均可能出现对寄存器新值的需求,所以判断时需要对各级分指令类型进行讨论.根据数据通路的不同,也即 Tuse/Tnew 在各级的取值,我们可以将指令分为 9 类(其中 nop 类是为了避免出现空指令时 type 保持不变而设置的).根据不同类型对 rs/rt 的 Tuse/Tnew 在 Tnew≤Tuse 时即可采取转发解决数据冲突.

3、转发的实现

首先明确,转发的源只能是流水寄存器或 RF 本身,否则会增大 CPU 关键路径的延时.所以转发可以通过 MUX 将各级的 V1/V2 传输到需要的端口处,当出现多级间的数据冒险时,选择离该级距离近的数据进行转发,因为最近的数据一定是最新的.所以,为转发单独设置一个控制器,对 D/E/M/W 级信号进行判断.其中,如果冲突发生在 W 和 D 之间,由于新值已经即将写入寄存器了,可以采用寄存器的内部转发实现,但是也可以采用 W 向 D 的转发从而简化 RF 内部的组合逻辑.同时需要注意,任何对 0 号寄存器的访问都只能转发 0 值,即使流水寄存器中一直保存 D 级读出的$0 的值,等价于让每一级流水 MUX 的选择信号都为 0.

附录：思考题及解答

Q: 请查阅相关资料,说一说什么是「FPGA 技术」？它有哪些好处和缺陷?

A:

FPGA（Field Programmable Gate Array）是在 PAL、GAL 等可编程器件的基础上进一步发展的产物.它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的,既解决了定制电路的不足,又克服了原有可编程器件门电路数有限的缺点.

缺点：成本高,资源贵,开发久且难.

优点：

① 既能管理又能运算;

② 功耗相对较低

③ 实时性：处理速度快,流水线并行和数据并行.

Q: 简述你的 UART 中断实现方案.

A：

判断 UART 的读取状态 rs,当 rs 为 1 时,说明接收的数据有效,进而可以触发中断.