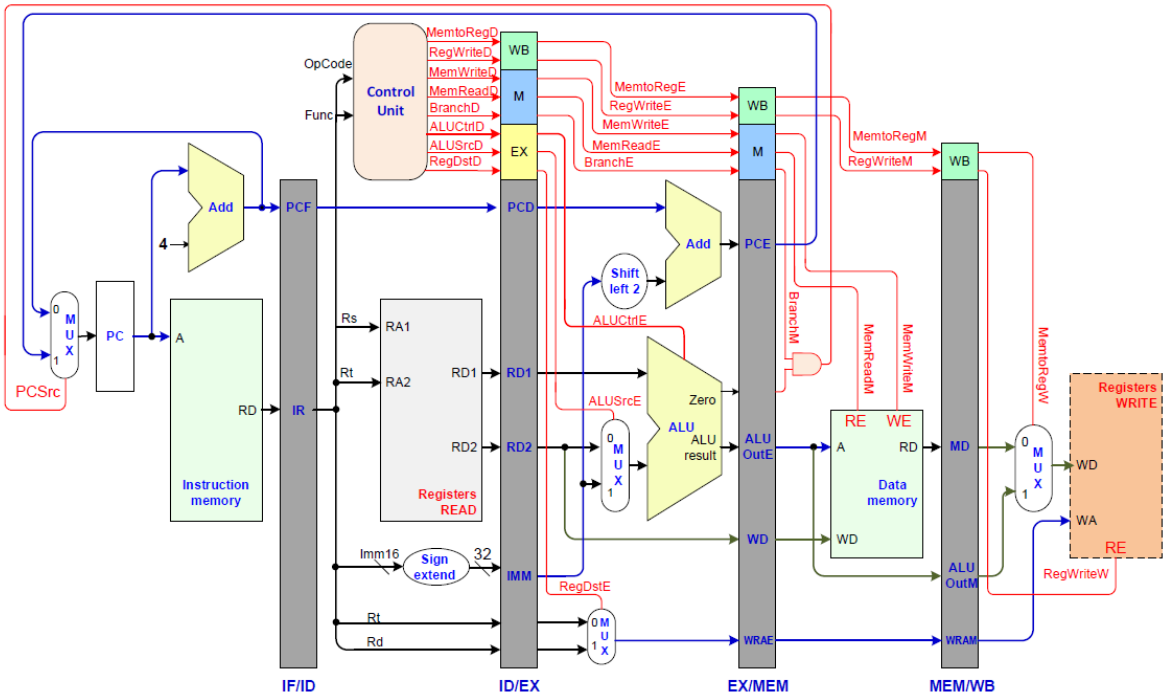
P5-Verilog流水线实验报告

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，支持的指令集包含{ addu, subu, ori, lw, sw, beq, lui, jal, jr, j, nop}。为了实现这些功能，CPU主要包含了IF，ID，EX，MEM，WB五级流水线及期间的寄存器，GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、EXT（位扩展器）等基本部件。这些模块按照mips-五级流水-部件的顶层设计逐级展开。



MIPS流水线数据通路（含控制信号）

（二）关键模块定义-流水线

> 命名规则：信号名 _ 流水线位置 _ I/O
>
> 其中，五级流水线分别为0，1，2，3，4

流水线级	代表序号
IF	0
ID	1
EX	2
MEM	3
WB	4

> 流水线之间的寄存器分别为01，12，23，34

流水线寄存器	代表序号
IF/ID	01
ID/EX	12
EX/MEM	23
MEM/WB	34

IF

取指令

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
NPC	I	下一个PC地址
PC	O	下一个PC地址
cmd	O	指令

IF/ID

级间寄存器

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
en01	I	是否可写
instr_01_I	I	指令
instr_01_O	O	指令
PC_01_I	I	PC
PC_01_O	O	PC

ID

- 解析IF/ID传来的指令
- 实例化1个控制器，取出RegWrite，RegDst，toReg信号向后传送。
- 实例化GRF，取寄存器的值
- 其中RegDstout, toRegout来自于WB级
- 来自RegDst, toReg选择器结果
- 实例化CMP，IFU，计算NPC

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr_1_I	I	指令
RegDstout_1_I	I	寄存器选择结果(WB)
toRegout_1_I	I	写入寄存器来源(WB)
RegWrite	I	写入寄存器信号
op_1	O	op
func_1	O	func
NPC_1	O	NPC
regRD1_1	O	regRD1
regRD2_1	O	regRD2
regWA_1	O	regWA
EXTout	O	扩展结果

ID/EX

级间寄存器，产生EX的控制信号

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
en12	I	是否可写
op_12_I	I	op
op_12_O	O	op
func_12_I	I	func
func_12_O	O	func
PC_12_I	I	PC
PC_12_O	O	PC
regWA_12_I	I	regWA
regWA_12_O	O	regWA
regRD1_12_I	I	regRD1
regRD1_12_O	O	regRD1
regRD2_12_I	I	regRD2
regRD2_12_O	O	regRD2
imm_12_I	I	拓展后的imm16
imm_12_O	O	拓展后的imm16

EX

使用ALU模块进行运算
实例化控制器生成信号ALU第二个操作数控制信号ALUSrc, ALU运算符控制信号ALU

信号名	方向	描述
regRD1_2	I	regRD1
regRD2_2	I	regRD2
imm_2	I	拓展后的imm16
op_2	I	op
func_2	I	func
ALUout_2	O	ALU结果

EX/MEM

级间寄存器

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
en23	I	是否可写
op_23_I	I	op
op_23_O	O	op
func_23_I	I	func
func_23_O	O	func
PC_23_I	I	PC
PC_23_O	O	PC
regWA_23_I	I	regWA
regWA_23_O	O	regWA
ALUOut_23_I	I	ALUOut
ALUOut_23_O	O	ALUOut
memWD_23_I	I	memWD
memWD_23_O	O	memWD
regWD_23_I	I	regWD
regWD_23_O	O	regWD

MEM

直接使用DM模块

MEM/WB

级间寄存器

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
en34	I	是否可写
op_34_I	I	op
op_34_O	O	op
func_34_I	I	func
func_34_O	O	func
PC_34_I	I	PC
PC_34_O	O	PC
regWA_34_I	I	regWA
regWA_34_O	O	regWA
ALUOut_34_I	I	ALUOut
ALUOut_34_O	O	ALUOut
memRD_34_I	I	memRD
memRD_34_O	O	memRD

WB

- 其中RegDstout, toRegout来自RegDst, toReg选择器结果

（三）关键模块定义-其他部件

Controller

信号名	方向	位数	描述
op	I	6	指令的6位op部分
func	I	6	指令的6位func部分
RegDst	O	2	寄存器写入选择信号 00：写入0寄存器，01：使用rt写入 10：使用rd写入 11：写入31寄存器
RegWrite	O	1	寄存器写入信号
ALUSrc	O	1	ALU运算数选择信号 0：grf 1：imm
branch	O	2	PC变更选择信号 00：自增4 01：暂无 10：j/jal 11：jr
beq	O	1	是否beq指令
MemWrite	O	1	内存写入信号
toReg	O	2	写入寄存器信号 00：ALU结果 01：内存 10：PC+4
extsel	O	2	选择拓展信号
ALU	O	4	ALU操作信号

指令信号对应表

	RegDst	RegWrite	ALUSrc	branch	beq	MemWrite	toReg	extsel	ALU
addu	01	1	0	00	0	0	00	x	0000
subu	01	1	0	00	0	0	00	x	0001
ori	00	1	1	00	0	0	00	00	0010
lw	00	1	1	00	0	0	01	01	0000
sw	x	0	1	00	0	1	x	01	0000
beq	x	0	0	01	1	0	x	x	0001
lui	00	1	1	00	0	0	00	00	0100
j	x	0	x	10	0	0	x	x	x
jal	10	1	x	10	0	0	10	x	x
jr	x	0	x	11	0	0	x	x	x
jalr	01	1	x	11	0	0	10	x	x

IFU

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号
NPC	I	下一个PC地址
cmd	O	指令

GRF

信号名	方向	位数	描述
WPC	I	32	指令的储存地址
clk	I	1	时钟信号
reset	I	1	复位信号
we	I	1	可写入信号
RA1	I	5	读出地址1
RA2	I	5	读出地址2
WA	I	5	写入地址
WD	I	32	写入数据
RD1	O	32	读出数据1
RD2	O	32	读出数据2

DM

信号名	方向	位数	描述
clk	I	1	时钟信号
reset	I	1	复位信号
op_2	I	6	op
func_2	I	6	func
MemWrite	I	1	可写入信号
Address	I	32	地址
WD	I	32	写入数据
RD	O	32	读出数据

ALU

信号名	方向	位数	描述
In1	I	32	运算数1
In2	I	32	运算数2
OP	I	4	操作信号
Zero	I	1	运算结果是否为0
Out	I	32	运算结果

运算符对应表(四位操信号备用)

OP	操作
0	加
1	减
2	或
3	与
4	将in2左移16位（用于lui）

EXT

用于拓展信号16->32

00	无符号拓展
01	有符号拓展
10	加载到高位
11	符号拓展后左移2位

信号名	方向	描述
Imm	I	16位立即数
EOp	I	操作符
ext	O	拓展后32位数

CMP

信号名	方向	描述
A	I	A
B	I	B
equal	O	A=B
bigger	O	A>B
smaller	O	A<B

（四）重要机制实现方法

1. 跳转

NPC模块和ALU模块协同工作支持指令x的跳转机制。

NPC模块内置了判定单元和计算单元来独立支持指令y的跳转机制。

2. 流水线延迟槽

无论如何都执行跳转指令的下一条指令

3. 转发

在Hazard模块中判断

CMP和jr使用的寄存器地址分别和EX，MEM，WB级将要写入地址作比较

ALU两个输入使用的寄存器地址分别和MEM，WB级将要写入地址作比较

DM的写入数据的寄存器地址与WB级将要写入地址作比较

产生控制信号，通过MUX进行转发

4. 进入中断处理程序

D级产生Tnew和Tuse，并与后续流水级进行比较，如果Tnew>Tuse则产生stall信号，冻结PC和IF/ID，将ID/EX清零

二、测试方案

（一）典型测试样例

1. 转发测试

```
ori $ra,$0,0x3014
jr $ra
ori $a0,$0,1
ori $a0,$0,2
ori $a0,$0,3
ori $a0,$0,4
```

```
45@00003000: $31 <= 00003014
75@00003008: $ 4 <= 00000001
95@00003014: $ 4 <= 00000004
```

2. 暂停测试

简单循环测试（需要暂停）

```
ori $t1,3
ori $t1,5
ori $t1,7
ori $t0,1
loop:
beq $t1,$s0,loop_end
nop
addu $s0,$s0,$t0
j loop
nop
loop_end:
```

结果

```
@00003000: $ 9 <= 00000003
@00003004: $ 9 <= 00000007
@00003008: $ 9 <= 00000007
@0000300c: $ 8 <= 00000001
@00003018: $16 <= 00000001
@00003018: $16 <= 00000002
@00003018: $16 <= 00000003
@00003018: $16 <= 00000004
@00003018: $16 <= 00000005
@00003018: $16 <= 00000006
@00003018: $16 <= 00000007
```

```
18@00003000: $ 9 <= 00000003
26@00003004: $ 9 <= 00000007
34@00003008: $ 9 <= 00000007
38@0000300c: $ 8 <= 00000001
50@00003018: $16 <= 00000001
70@00003018: $16 <= 00000002
90@00003018: $16 <= 00000003
110@00003018: $16 <= 00000004
130@00003018: $16 <= 00000005
150@00003018: $16 <= 00000006
170@00003018: $16 <= 00000007
```

（二）自动测试工具

1. 测试样例生成器

2. 自动执行脚本

3. 正确性判定脚本

三、思考题

流水线冒险

1. 在采用本节所述的控制冒险处理方式下，PC的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

   在D级解析指令，计算NPC，若无特殊控制信号，则NPC为PC+4，若指令为跳转指令，计算出相印的NPC，将此NPC反馈回IF。

2. 对于jal等需要将指令地址写入寄存器的指令，为什么需要回写PC+8？

   延迟槽的存在使得跳转指令的下一条将被执行，即应该记录PC下下条的指令。

数据冒险的分析

为什么所有的供给者都是存储了上一级传来的各种数据的**流水级寄存器**，而不是由ALU或者DM等部件来提供数据？

这些部件为组合逻辑，如果将结果转发回来可能会导致出现错误值或者发回不需要的值

AT法处理流水线数据冒险

1. “转发（旁路）机制的构造”中的Thinking 1-4；
   1. 如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题

      会导致使用的数据是未写入的数据。例如ori $t0, $0 ,1 后跟 addu $t1 $t0, $0 如果采用上一级原始数据会导致$t0为0，而实际上是1

   2. 我们为什么要对GPR采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

      为了在WB的数据尚未写入GPR时读取GPR能得到正确的值。可以采用clk下跳沿写入数据，即可读出稳定的数据，或者采取和其他流水线级一样的转发策略。

   3. 为什么0号寄存器需要特殊处理？

      0号寄存器需要始终保持为0，就算地址冲突了，也不可以将其他数据转发过来。

   4. 什么是“最新产生的数据”？

      距离需求者最近的寄存器的数据，距离较远的将被覆盖。

2. 在AT方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为0”，但在CPU写入GRF的时候，是有一个we信号来控制是否要写入的。为何在AT方法中不需要特判we呢？为了**用且仅用**A和T完成转发，在翻译出A的时候，要结合we做什么操作呢？

   即使该数据非需要写入寄存器的数据，在下一周期也会重新进行转发，进行转发也不会影响最终结果。结合WE判断是否会产生新的寄存器值。

在线测试相关说明

> 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

> 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略**，比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

> 此思考题请同学们结合自己测试CPU使用的具体手段，按照自己的实际情况进行回答。

手动构造样例。

连续暂停

```
ori $a0,$0,2026
sw $a0,0($0)
lw $a1,0($0)
lw $a2,0($0)
beq $a2,$a1,branch
ori $t0,$0,1
ori $t0,$0,2
branch:
ori $t1,$0,3
```

```
18@00003000: $ 4 <= 000007ea
18@00003004: *00000000 <= 000007ea
26@00003008: $ 5 <= 000007ea
30@0000300c: $ 6 <= 000007ea
46@00003014: $ 8 <= 00000001
50@0000301c: $ 9 <= 00000003
```