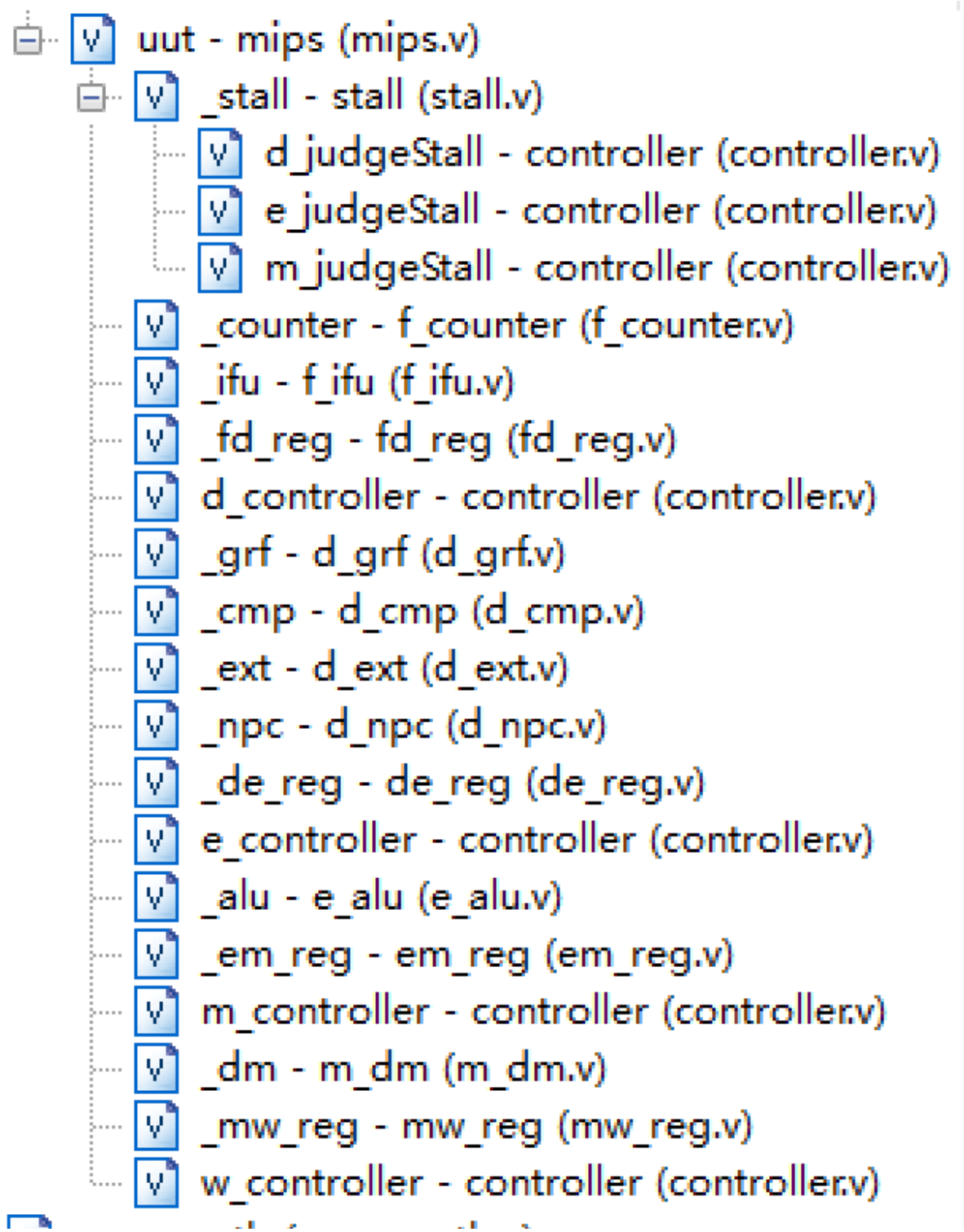
P5设计文档

命名规则：

1. 主模块为mips.v，命名为mips
2. 特殊模块为stall和controller，其中stall命名为_stall，controller在stall中命名为“流水线层级_judgeStall”，在mips中命名为“流水线层级_controller”
3. 流水线寄存器有四个，命名为“流水线连续层级_reg“
4. 其余模块命名为”_模块名“

H2 整体架构



总体设计思路参考了教程和一些学长的灵感

H2 controller

采用了分布式译码的写法，即完成一个大的controller，再将其实例化多次

下面对controller中的每一个信号做出讲解

(为了设计时方便起见，所有的信号除了写使能信号外，均设置为4位，其中若tuse的第3位置1，则为高阻态)

judgeStall

	add	sub	ori	lw	sw	beq	lui	jal	jr	sll	j	lh	sh	lb	sb
tuse_rs	1	1	1	1	1	0	\	\	0	\	\	1	1	1	1
tuse_rt	1	1	\	\	2	0	\	\	\	1	\	\	2	\	2
tnew	2	2	2	3	0	0	1	1	0	2	0	3	0	3	0

tuse_rs

在几个时钟周期到来后会使用grf_rs

\为高阻态

tuse_rt

在几个时钟周期到来后会使用grf_rt

\为高阻态

tnew

在几个时钟周期到来后能够将写入写入寄存器的内容写入流水寄存器

只有当tuse>=tnew时，电路才能正常流水，否则就要阻塞

---

d_controller

	add	sub	ori	lw	sw	beq	lui	jal	jr	sll	j	lh	sh	lb	sb
cmpOp	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
nPcOp	0	0	0	0	0	1	0	2	3	0	2	0	0	0	0
extOp	0	0	5	1	1	2	3	4	0	0	4	1	1	1	1

cmpOp

当分支判断语句到来时，控制cmp模块中进行运算的类型，输出branch

0	1
nope	=

nPcOp

控制nPc模块跳转pc的类型

0	1	2	3
pc4	pcImm16	pcImm26	pcReg

extOp

控制ext模块中拓展立即数的类型，输出extImm

0	1	2	3	4	5
nope	signext	signext00	sll16	ext00	zeroext

注意：

1. signex00为16位offset的拓展，结果已经加上了pc+4
2. ex00为26为立即数的拓展，结果已经在前面复制了四位pc

---

e_controller

	add	sub	ori	lw	sw	beq	lui	jal	jr	sll	j	lh	sh	lb	sb
srcASel	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1
srcBSel	1	1	2	2	2	0	0	0	0	1	0	2	2	2	2
aluOp	1	2	3	1	1	0	0	0	0	4	0	1	1	1	1

srcASel

控制alu中srcA的类型

0	1
nope	grf_rs

srcBSel

控制alu中srcB的类型

0	1	2
nope	grf_rt	extImm

aluOp

控制alu中进行运算的类型

0	1	2	3	4
nope	+	-	|	<<

其中<<为sll语句，lui在d级就已经被执行了

---

m_controller

	add	sub	ori	lw	sw	beq	lui	jal	jr	sll	j	lh	sh	lb	sb
memWrite	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
memOp	0	0	0	1	1	0	0	0	0	0	0	2	2	3	3

memWrite

dm写使能信号，1为写，0为不写

memOp

控制当前对dm进行读写操作的数据类型

0	1		3
nope	w	h	b

转发相关信号

	add	sub	ori	lw	sw	beq	lui	jal	jr	sll	j	lh	sh	lb	sb
regDstSel	1	1	2	2	0	0	2	3	0	1	0	2	0	2	0

	add	sub	ori	lw	sw	beq	lui	jal	jr	sll	j	lh	sh	lb	sb
regWdSel	1	1	1	2	0	0	3	4	0	1	0	2	0	2	0

regDstSel

选择写入寄存器的位置来源

0	1	2	3
0	[15:11]	[20:16]	31

注意：

1. 在controller中最后会直接输出regDst，而非选择信号
2. 当写入位置为0时，视为regWrite为0

regWdSel

选择写入寄存器的内容来源

0	1	2	3	4
nope	aluResult	memRd	extImm	pc4

唯一需要注意的是pc4在转发和写入时其实为pc+8，这是由延迟槽的性质决定的

测试思路

1. 随机出现指令，指令为[add,sub,beq,lui,jal,jr,j,ori,sll,lw,sw]中的一种
2. 使用c++编写单周期程序，保证测试程序合理性
3. 不断生成，以得到更到强度的程序
4. 对于剩余没有覆盖到的点，自己手动构造数据进行测试

自己跑出的一个点

```
standard pipeline-cycle: 879
slow pipeline-cycle: 1695
accepted cycle range: [715, 1450]
```

在自己构造数据测试时，我发现了cpu对beq在阻塞中的行为在一些特殊情况下有异常，并成功通过改变阻塞的位置解决的问题

思考题

1、我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

```
1 ori $t0,$0,10
2 beq $t0,$t0,label
3 nop
```

在这个例子中，因为ori在d级时的tnew为2，造成了冒险，所以程序会让流水线阻塞；若beq下放到e级判断，则程序就可以不阻塞地完成任务

2、因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计？

因为跳转指令只有当来到D级时才能确定是否跳转，此时下一条语句已经进入了F级，所以实际跳转时将会跳转当前D级PC+8的地址

3、我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM 、 ALU ），请思考为什么？

1. 流水寄存器的输出是稳定的，只会在时钟上升沿进行改变；而功能部件的输出为组合逻辑，会随着毛刺产生各种不同的值，可能导致电路产生意想不到的后果
2. 在转发时，势必要用到多个功能部件，如果每个功能部件都进行转发，会导致转发过程极为散乱，影响cpu设计

4、我们为什么要使用 GPR 内部转发？该如何实现？

当我们要对一个寄存器又读又写时，会造成冒险冲突（尤其是写后读）。此时，内部转发可以一方面在不影响写入的情况下获得正确的数据，另一方面可以简化布线。具体可以通过在grf里用组合逻辑对输出赋值实现。

5、我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路？

每一级的数据只可能来自它的后方

D<– E M W

E<– M W

M<– W

其中，每一级转发的数据只可能来自“aluResult,memRd,extImm,pc4”四部分，所以理论上有24条数据通路

6、在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

- R型/I型运算类（运算）

  增加alu指令，较简单

- B+alu类（条件跳转）

  添加流水信号，判断寄存器写入位置

- lw/sw + [condition]类（条件访存）

  添加流水信号，同时在dm中添加组合逻辑访存

7、确定你的译码方式，简要描述你的译码器架构，并思考该架构的优势以及不足。

我的译码方式为两重与逻辑+一重或逻辑

1. 决定op对应的指令，若为000000则special置一
2. 若special为1，则根据funct决定指令
3. 根据指令得到不同的信号

其中，我信号获取时采用的是按位或的方式，这样做的好处是添加新指令非常方便，缺点是需要自己对信号进行二进制译码，容易出错