使用verilog的五级流水线MIPS CPU设计文档

0. 综述

本CPU为Logisim实现的单周期CPU，支持的指令集包含`{addu,subu,ori,lw,sw,lb,sb,lui,nop,beq,j,jal,jr}`。为了实现这些功能，CPU主要包含了：IFU、GRF、ALU、DM、Ext、Control、BrCmp、Datapath、MUX_4、NPC、settings等模块。

- stage 部分模块：D_part模块（GRF、Ext、NPCcalu、BrCmp、Trans_grf_mux1、Trans_grf_mux2）；Ex_part模块（ALU、Trans_ALUIn_MUX1、Trans_ALUIn_MUX2）；MEM_part（Dm、Trans_MemRD_MUX）；WB_part；
- 流水线寄存器模块：IFtoIDreg、IDtoEXreg、EXtoMEMreg、MEMtoWBreg等模块

一、模块规格

1. F_part

1.1 IFU模块

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
NPC	Input	[31:0]	下一次PC值
Instruction	Output	[31:0]	当前指令
PCOut	Output	[31:0]	当前PC值

具体功能：

功能	描述
输出下一条指令	在clk上升沿时，① 当PCSrc为0时，$PC \leftarrow PC+4$; ② 当PCSrc为1时，$PC \leftarrow PC+4+sign\_ext(offset\|\|0^2)$
输出当前PC值	
复位	reset信号变为1时，PC清零

1to2. IFtoIDreg

2.D_part

2.1 GRF模块

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
WE	Input	1	写使能端
ReadAddr1	Input	[4:0]	读寄存器编号1
ReadAddr2	Input	[4:0]	读寄存器编号2
WriteAddr	Input	[4:0]	写寄存器编号
WData	Input	[31:0]	写入数据
RData1	Output	[31:0]	读寄存器值1
RData2	Output	[31:0]	读寄存器值2

具体功能：

功能	描述
复位	当reset为1时，所有寄存器的值清零
读取数据	RData1的值是寄存器编号为Read1的寄存器的值；RData2的值是寄存器编号为Read2的寄存器的值
写入数据	当写使能WE为1时，向编号为Write的寄存器写入WriteData

2.2 Ext模块

端口定义：

端口名	方向	位宽	功能描述
Imm16	Input	[15:0]	输入的16位立即数
EXtCtrl	Input	[1:0]	控制信号
Imm32	Output	[31:0]	扩展结果

具体功能：

功能	描述
0扩展	`Imm32={{16{0}},Imm16}`
符号扩展	`Imm32={{16{Imm16[15]}},Imm16}`
把数加载到高位	`Imm32={Imm16,{16{0}}}`
1扩展	`Imm32={{16{1}},Imm16}

2.3 NPC

端口名	方向	位宽	功能描述
ifBr	I	1	是否满足B类跳转条件
isBr	I	1	是否是Branch指令
isJump	I	1	是否是J类型指令
isJr	I	1	是否是Jr类型指令
BrImm	I	[31:0]	Br类型跳转立即数
JImm	I	[31:0]	J类型跳转立即数
JrImm	I	[31:0]	Jr类型跳转立即数
PC	I	[31:0]	PC值
NPC	O	[31:0]	Next PC值

2.4 BrCmp

端口名	方向	位宽	功能描述
Instr	I	[31:0]	指令
RData1	I	[31:0]	读入数据1
RData2	I	[31:0]	读入数据2
ifBr	O	1	是否满足B类跳转条件

2to3. IDtoEXreg

3. Ex_part

3.1 ALU

端口定义：

端口名	方向	位宽	功能描述
A	Input	[31:0]	输入数据1
B	Input	[31:0]	输入数据2
ALUOp	Input	[4:0]	选择ALU功能
Shamt	Input	[4:0]	左移位数
Zero	Output	1	运算结果是否为0
Result	Output	[31:0]	运算结果

具体功能：

功能	描述
与	`Result=A&B`
或	`Result=A|B`
异或	`Result=A^B`
减（比较）	`Result=A-B`；若 `A==B,Zero=0`
加	`Result=A+B`
逻辑左移	`Result=B<<Shamt`

3to4. EXtoMEMreg

4. MEM_part

4.1 Dm

端口定义：

端口名	方向	位宽	功能描述
Clk	Input	1	时钟信号
Reset	Input	1	异步复位信号
WE	Input	1	写使能信号
isMemb	I	1	对内存操作按字节
isMemh	I	1	对内存操作按半字
Addr	Input	[4:0]	写入地址
WD	Input	[31:0]	写入数据
RD	Output	[31:0]	读取数据

具体功能：

功能	描述
复位	Reset 为 1 时，数据清 0
写入数据	当时钟上升沿时，如果写使能信号WE有效，就将WD写入到地址为Addr处
读取数据	RD为地址Addr处的数据的值

4to5. MEMtoWBreg

5. WB_part

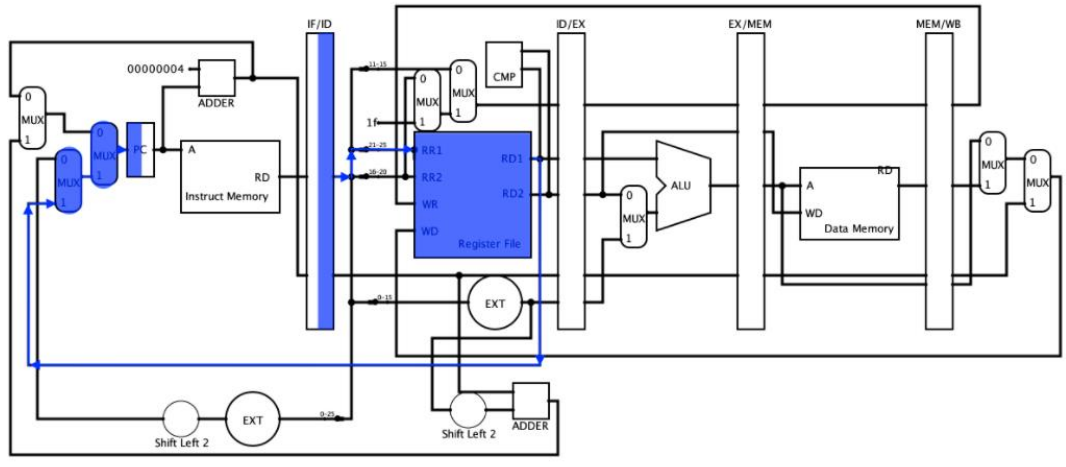
回写。

6. datapath

将前述5个模块实例化并连接

端口名	方向	位宽	功能描述
clk	In	1	时钟信号
reset	In	1	异步复位信号
Trans_grf_Sel1	In		转发更新RD1的MUX控制信号
Trans_grf_Sel2	In		转发更新RD2的MUX控制信号
Trans_ALUIn_Sel1	In		转发更新E_RD1的MUX控制信号
Trans_ALUIn_Sel2	In		转发更新E_RD2的MUX控制信号
Trans_MemRD_Sel	In		转发更新M_RD2的MUX控制信号
stall	In		暂停信号
Tuse1	Out		D级指令的Tuse1，用于暂停判断
Tuse2	Out		D级指令的Tuse2，用于暂停判断
D_ReadA1	Out		D级指令要读取的寄存器1地址
D_ReadA2	Out		D级指令要读取的寄存器2地址
E_ReadA1	Out		E级指令要读取的寄存器1地址
E_ReadA2	Out		E级指令要读取的寄存器2地址
E_WriteA	Out		E级指令写入寄存器的地址
E_Tnew	Out		E级指令的Tnew
M_ReadA2	Out		M级指令要读取的寄存器2地址
M_WriteA	Out		M级指令写入寄存器的地址
M_Tnew	Out		M级指令的Tnew
W_WriteA	Out		W级指令写入寄存器的地址
W_Tnew	Out		W级指令的Tnew

1. ID



二、控制信号

1. 指令编码

	addu	subu	ori	lw	sw	beq	lui	sll
op	000000	000000	001101	100011	101011	000100	001111	000000
func	100001	100011	n/a	n/a	n/a	n/a	n/a	000000
	j	jal	jr					
op	000010	000011	000000					
func	n/a	n/a	001000					

2. 主控制信号真值表

	RegDst	RegWrite	EXTCtrl	ALUSrc	ALUOp	WE	MemToReg	isBr	isJump	isJR
作用	NULL	GRF写使能	扩展器控制	MUX选择ALU端口B的输入	ALU功能	DM写使能	MUX选择写回数据（有修改）	是否是beq指令	是否J类跳转	是否JR
addu	1	1	x	0	add	0	0	0	0	0
subu	1	1	x	0	sub	0	0	0	0	0
ori	0	1	00	1	or	0	0	0	0	0
lw	0	1	01	1	add	0	1	0	0	0
sw	x	0	01	1	add	1	x	0	0	0
beq	x	0	x	0	sub	0	x	1	0	0
lui	0	1	10	1	add	0	0	0	0	0
nop	x	0	x	x	x	0	x	0	0	0
sll	1	1	x	0	sll	0	0	0	0	0
j	x	0	sign	1	add	0	x	0	1	0
jal	恒31	1	x	x	x	0	10	0	1	0
jr	x	0	x	0	x	0	0	0	0	1

	0	1	2	3
RegDst	Rt	Rd	31	x

注：在本人目前的设计中，写入地址在ATControl中产生，因此RegDst暂无用处。

	0	1	2	3
ExtCtrl	无符号扩展	符号扩展	加载到高位	1扩展

	0	1	2	3
MemtoReg	ALU结果	DM结果	PC+8	x

update：为将jal一类指令的Tnew缩短，本人将PC+8提前到了D_part，通过ImmSel选择输入ALU的立即数，ALU方法为加载B口数据，以及注意搭配ALUSrc使用。

3. AT信号译码

Tuse：这条指令位于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。

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。

	转发接受位点	Tuse	转发输出位点	Tnew(D_part)	WriteAddr
addu|subu	E@ALUIn1&In2	rs=rt=1	EMreg	2	rd
sll	E@ALUIn2	rt=1	EMreg	2	rd
jr	D@jrImm(RD1)	rs=0		`n/a`	
lui		`n/a`	DEreg	1	rt
ori	E@ALUIn1	rs=1	EMreg	2	rt
lw	E@ALUIn1	rs=1	MWreg	3	rt
sw	E@ALU M@DmIn	rs=1,rt=2		`n/a`	
beq	D@cmpIn1&In2	rs=rt=0		`n/a`	
j		`n/a`		`n/a`	
jal		`n/a`	DEreg	1	ra

4. 转发控制

需要设置五个MUX

1. 更新RData1

$$MUX1\begin{cases}GRFRdata1\\DE\_reg转发过来的数据\\EM\_reg转发过来的数据\\MW\_reg转发过来的数据\end{cases}$$

MUX控制信号生成：

$$if\ 读取地址为0,Sel=0$$
$$else\ if\ 读取地址与E级要写入地址相同，且E级写使能为1且此时要写入数据已产生,Sel=1$$
$$else\ if\ ...M...,Sel=2$$
$$else\ if\ ...W...,Sel=3<就近原则>$$
$$else\ 没有需要转发的数据,Sel=0$$

2. 更新RData2

   与更新RData1类似

3. 更新经DEreg流水线寄存器流水过来的E_RData1

$$E\_Trans\_RData1\begin{cases}E\_RData1\\EM\_reg转发过来的数据(E级的ALUResult存到reg中的数据)\\MW\_reg转发过来的数据(M级选择好的要写回GRF的数据，存到reg中)\end{cases}$$

[注：因为在D级更新RData1时，后续的阶段可能尚未计算出转发的结果，因此需要在E级继续转发更新]

MUX控制信号生成：

$$
\begin{gathered}
if\ 读取地址为0, Sel = 0 \\
else\ if\ 读取地址与M级要写入地址相同，且M级写使能为1且此时要写入数据已产生, Sel = 1 \\
else\ if\ ...W..., Sel = 2 < 就近原则 > \\
else\ 没有需要转发的数据, Sel = 0
\end{gathered}
$$

4. 更新经DEreg流水线寄存器流水过来的E_RData2

   与E_RData1类似

   注意向后流水的时候是更新后的E_Data2(E_Trans_RData2)

5. 更新写入dm的数据

$$
MemWData \begin{cases} GRFRData2(E\_Trans\_RData2经EM寄存器流过来的数据) \\ W\_GRFWData \end{cases}
$$

MUX控制信号生成：

$$
\begin{gathered}
if\ 读取地址为0, Sel = 0 \\
else\ if\ 读取地址与W级要写入地址相同，且W级写使能为1且此时要写入数据已产生, Sel = 1 \\
else\ 没有需要转发的数据, Sel = 0
\end{gathered}
$$

5. 阻塞控制

为了方便处理，下述暂停是指将指令暂停在D级。暂停操作：

- 冻结PC的值
- 冻结F/D级流水线寄存器的值
- 将D/E级流水线寄存器清零（这等价于插入了一个nop指令）

当D级指令读取寄存器的地址与E级或M级的指令写入寄存器的地址相等且不为0，且D级指令的Tuse小于对应E级或M级指令的Tnew时，我们就需要在D级暂停指令。在其他情况下，数据冒险均可通过转发机制解决。

$$
\begin{gathered}
if\ 当前处于E级的指令与D级指令构成转发关系(写后读), Tnew1 = E\_Tnew \\
else\ if\ ...D..., Tnew1 = D\_Tnew \\
else\ if\ W... \quad [Tnew1是rs对应的Tnew] \\
Tnew2同理;
\end{gathered}
$$

阻塞信号：`stall=(D_ReadA1!=0 && Tnew1>Tuse1) || (D_ReadA2!=0 && Tnew2>Tuse2)`

三、思考题

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。

   数据通路：数据来源为当前IFU的PC值，D级的B类型立即数（$sign\_extend(offset||0^2)$）、J类型立即数($PC_{31..28}||instr\_index||0^2$)、jr类型立即数（$R[rs]$）

   控制信号：根据`MUXSrc`选择立即数，根据`isJr,isJump,ifBr,isBr`产生`PCSrc`。注意因为这里的Imm是D级产生的，而D级的PC=F级PC-4，所以对于B类跳转指令，
   $NPC = PC_B + Imm + 4 = PC_F + Imm$

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？

   若要将暂停取指时间利用起来：不论判断结果如何，我们都将执行分支或跳转指令的下一条指令。即“延迟槽”。编译器调度会适当的指令移入延迟槽中，充分利用流水线的性能。因此PC+4的指令是延迟槽中的指令，PC+8才是程序中看到的jal的下一条指令的PC值。

3. 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？

这样做功能虽正确，会导致流水线各阶段延迟不均衡，CPU时钟频率大幅度降低，从而使得流水线性能严重下降。

4. 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。

   会导致数据因为没有及时更新而出错。

5. 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？

   因为GRF既可以视为D级的一个部件，也可以视为W级之后的流水线寄存器。从而可以在D级读寄存器时读取到WB阶段写入GRF的数据，也相当于转发，只不过设置在GRF内部，且实现过程更加简洁。不采用内部转发，需要另开辟线路转发该数据。

6. 为什么0号寄存器需要特殊处理？

   对0号寄存器写入不会改变0号寄存器的值，若读取时不特判零，就会在读取0号寄存器的值时，将写入的数据读出，而不是读出0号寄存器真正的值(0)

7. 什么是“最新产生的数据”？

   即在指令顺序执行时，距离当前指令更近的前序指令。因为对同一寄存器写入，相距更近的指令会覆盖之前的指令写入的值，因而在有多个转发输入来源都满足条件时，应当按E、M、W的优先级来转发。

8. 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？

   在翻译A时，将不写寄存器的writeA置为0，这样其实相当于不写入数据。因此便无需特判WE。

9. 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？

   写后读会产生冲突，所以可以根据这一点构造测试数据。

   需要写入寄存器的指令有`addu(rd),subu(rd),jalr(rd),lui(rt),ori(rt),lw(rt),jal(ra)`；需要读取寄存器的指令有`addu|subu(rs,rt),Jr(rs),jalr(rs),ori(rs),lw(rs),sw(rs,rt),beq(rs,rt)`。所以可以根据这些指令经排列组合构造冲突情况。

```
ori $s0,4
    ori $s1,10
    ori $s2,1
    addu $a0,$0,$s1
    jal func
    jal end

func:
    sw $ra,0($t1)
    addu $t1,$t1,$s0

    beq $a0,$s2,func_ret
        sw $a0,0($t1)
        addu $t1,$t1,$s0

        subu $a0,$a0,$s2
        jal func

        subu $t1,$t1,$s0
        lw $a0,0($t1)
```

```
        addu $v0,$v0,$a0

        jal func_end

func_ret:
    addu $v0,$0,$s2
    j func_end

func_end:
    subu $t1,$t1,$s0
    lw $ra,0($t1)
    jr $ra
end:
    lui $t7,16
```

```
# test ori
ori $a0, $0, 123
ori $a1, $a0, 456

# test lui
lui $a2, 123
lui $a3, 0xffff
ori $a3, $a3, 0xffff    # $a3 = -1

#test addu
addu    $s0, $a0, $a2   # ++
addu    $s1, $a0, $a3   # +-
addu    $s2, $a3, $a3   # --

# test sw
ori $t0, $0, 0x0000
sw  $a0, 0($t0)
sw  $a1, 4($t0)
sw  $a2, 8($t0)
sw  $a3, 12($t0)
sw  $s0, 16($t0)
sw  $s1, 20($t0)
sw  $s2, 24($t0)

# test lw
lw  $a0, 0($t0)
lw  $a1, 12($t0)
sw  $a0, 28($t0)
sw  $a1, 32($t0)

# test beq
ne:
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, ne
beq $a0, $a2, eq


eq:
sw  $a1, 40($t0)
```

```
# test sll,j
ori $t0, $0, 3
nop
test:
sll $t0, $t0, 2
nop
sll $t1, $t0, 2
nop
j   test
```

```
ori $t1, $0, 32
ori $t2, $0, 0
ori $t0, $0, 0
nop
nop
for_begin:
beq $t0, $t1, for_end
sw  $t0, 0($t2)
ori $s0, $0, 4
addu    $t2, $t2, $s0
ori $s0, $0, 1
addu    $t0, $s0, $t0
beq $0, $0, for_begin
for_end:
nop
nop
```

```
ori     $t0, $zero, 1   # t0=1
ori     $t1, $zero, 2   # t1=2
ori     $t3, $zero, 4   # t3=4
ori     $t2, $t0, 1 # t2=1
sw      $t2, 8($zero)
sw      $t1, 12($zero)
lw      $t2, 8($t3) # t2=2
eq:
lw      $t1, 8($zero)   # t1=1
addu        $t4, $t1, $t0   # t4=2
addu        $1, $1, $t1 # $1=1
sw          $1, 4($zero)
lw          $2, 4($zero)    # $2=1
beq     $t4, $t0, eq    # not equal
subu        $t4, $t4, 1 # t4=1
j       eq
beq     $t4, $t0, eq    # equal
```

```
ori $15 $0 56
ori $16 $0 60
ori $17 $0 84
ori $18 $0 96
ori $19 $0 128
ori $20 $0 12
ori $21 $0 12
ori $22 $0 12
ori $23 $0 24
ori $24 $0 56
```

```
sw $16 0($15)
sw $17 0($16)
sw $18 0($17)
sw $19 0($18)
sw $20 0($19)
sw $21 0($20)
sw $22 0($21)
sw $23 0($22)
sw $24 0($23)
sw $25 0($24)
lw $15 0($15)
beq $7 $15 beq_0
addu $5 $7 $22
beq_0:
lw $16 0($16)
beq $17 $16 beq_1
addu $11 $17 $3
beq_1:
lw $17 0($17)
beq $16 $17 beq_2
addu $11 $16 $12
beq_2:
lw $18 0($18)
beq $17 $18 beq_3
addu $5 $17 $16
```