使用verilog的五级流水线MIPS CPU设计文档

0. 综述

本CPU为erilog实现的五级流水线多周期CPU，支持的指令集包含{LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、 SUBU、 MULT、 MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO} + {MOVZ、RtBgezal}。为了实现这些指令，CPU主要包含了：IFU、GRF、ALU、MDU、DM、Ext、Control、TransStallController、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、MDU、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

具体功能：

计算下一条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

2.5 CdtWECmp

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	I	1	是否满足B类跳转条件
ifCdtWE	O	1	是否满足条件写的条件

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`

3.2 MDU

端口定义：

端口名	方向	位宽	功能描述
clk	In	1	时钟信号
reset	In	1	复位信号
A	In	[31:0]	输入数据1
B	In	[31:0]	输入数据2
MDUOp	In	[4:0]	乘除模块运算类型
MDUResult	Out	[31:0]	MDU结果
Busy	Out	1	模拟延迟

具体功能：

执行`MULT、MULTU、DIV、DIVU、MTLO、MTHI`指令的乘除部分要求

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

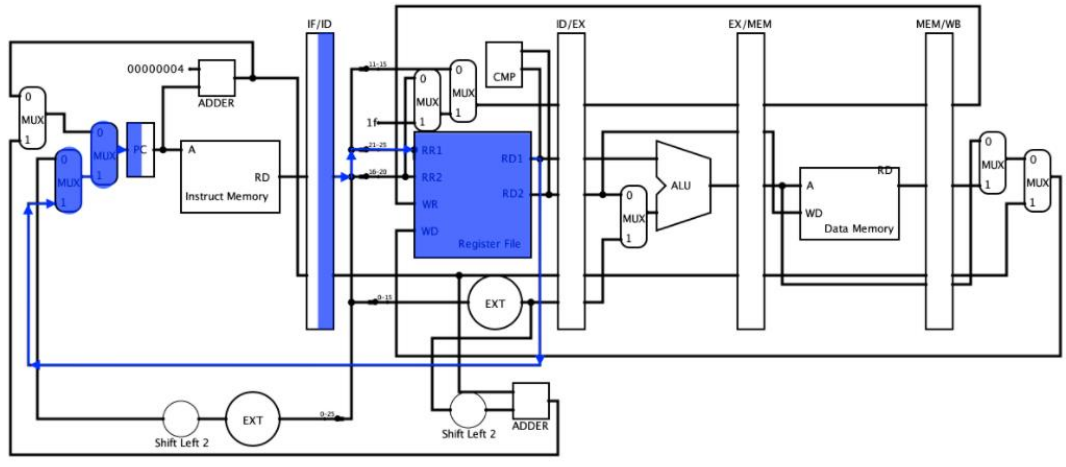
回写。

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

	RegWrite	EXTCtrl	ALUSrc	ALUOp	WE	MemToReg	isBr	isJump	isJR	MemType	MemSign	MDUStart	MDUOp	EResultSel	ConditionWE
作用	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	对Mem的操作位宽	对Mem符号操作类型	MDU模块运算启动信号	MDU模块运算类型	选择ALU的MDU的运算结果	是否为条件写类型
addu	1	x	0	add	0	0	0	0	0						
subu	1	x	0	sub	0	0	0	0	0						
ori	1	00	1	or	0	0	0	0	0						
lw	1	01	1	add	0	1	0	0	0	word	Unsigned				
sw	0	01	1	add	1	x	0	0	0	word	Unsigned				
beq	0	x	0	sub	0	x	1	0	0						
lui	1	10	1	add	0	0	0	0	0						
nop	0	x	x	x	0	x	0	0	0						
sll	1	x	0	sll	0	0	0	0	0						
j	0	sign	1	add	0	x	0	1	0						
jal	1	x	x	x	0	10	0	1	0						
jr	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	x	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
mult	E@MDUIn1&In2	rs=rt=1			
mfhi			EMreg	2	rd
mthi	E@MDUIn1	rs=1			
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

类`addu`：`add,sub,ans,or,xor,nor,sllv,srlv,srav,slt,sltu`;

类`sll`：`srl,sra`;

类`mult`：`multu,div,divu`;

类`mfhi`：`mflo`;

类`mthi`：`mtlo`;

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

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

[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：

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

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

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

MUX控制信号生成：

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

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。

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

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

三、思考题

1. 为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI、LO寄存器？

   乘除法运算有延迟，执行时间并非单周期。如果整合进ALU，会导致在ALU进行乘除指令的运算时，其他需要用到ALU的指令都需要暂停等待，造成很大的时间浪费。分离单独的乘除法部件可以在计算运行乘除结果时运行其他ALU指令，从而提高效率。

   设置独立的HI、LO寄存器的原因有：①用于存储不适合单个寄存器的操作结果。因为32位×32位的运算可以得到64位的结果，32位除32位需要分别保存商和余数。MIPS的指令设计类型中，R型只指定一个寄存器。因此需要另设两个独立寄存器用于保存乘除结果。②乘除模块运算得到结果需要时间超过一周期，单独设置存储数据的寄存器可使逻辑更清晰。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

   延迟槽即位于分支跳转指令之后的一条指令，不管是否需要分支跳转，这条指令总是被执行。因为分支跳转指令需要到D级才能判断对PC产生何种影响。

   而乘除槽指的是在乘除法运算指令后需要用到MDU的指令由于乘除法运算的缓慢需要被延迟。而不用到MDU的指令继续执行，与延迟槽类似。

3. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑C语言中字符串的情况）

   当对内存的读取操作数据位宽小于一字时，性能更有优势。因为C语言字符串每个字符都是一字节，所以在这种情景下，按字节读取性能更有优势。

4. 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?

   因为只有读后写才需要解决冲突，因此我们将写寄存器的Tnew分类，读寄存器的Tuse分类，通过对两者进行枚举与排列组合，我们可有构造出所有的冲突类型。

Pattern	Instr
rd, rs, rt	add(u),sub(u),and,or,slt(u),sllv,srlv,srav
$rd, rt, shamt$	sll,srl,sra
rt, rs, Imm	ori,addi(u),andi,xori,lui,slti(u)
$s*\ rt, offset(rs)$	sw,sb,sh
$l*\ rt, offset(rs)$	lw,lb(u),lh(u)
$rs, rt, offest$	beq,bne
$rs, offest$	bgtz,bltz,blez,bgez
j addr	j
jal addr ra	jal
rs	jr
rd, rs	jalr
rs	mthi,mtlo
rd	mfhi,mflo
rs, rt	mult(u),div(u)

5. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助?

   1. 采用特定的信号来判断特定的功能。对于转发和阻塞机制，我们将指令翻译出的Addr和Tuse、Tnew信号用于判断，而无需用其他纷杂的信号处理。

      AT法方便直观，且具有普适规律。对于一类的信号，它们的AT信号常常相同，这样就可以大大简化我们的工作量。

   2. 对指令的Op、Funct、Rt等类型以及一些变化较多的信号用宏定义，这样清楚直观，不易出错。

部分测试数据：

```
    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
```

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

```mipos
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
```
```

```mips
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
```

```mips
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
```

```
   li $12,0x309c0
   srl $4,$12,4
   jalr $10,$4
   xori $1,$1,1
   li $12,0x30b00
   sra $4,$12,4
   jalr $10,$4
   xori $1,$1,1
   li $15,0x313c
   li $16,0x1
   mult $15,$16
   mflo $4
   jalr $10,$4
```

```
   li $3,0xfedcba98
   li $4,0

   sw $3,0($4)
   addiu $3,$3,0x01010101
   addu $4,$4,4
   lbu $3,0($0)
   add $3,$3,$3
   lw $3,4($0)
   addiu $4,$3,0x10101010
   slt $5,$3,$4
   addiu $3,$3,2
   slt $5,$3,$4
   lw $3,4($0)
   slt $3,$3,$4
   lw $3,4($0)
   sra $3,$3,1
   lw $3,4($0)
   sra $3,$3,2
   lw $3,4($0)
   sra $3,$3,3
```

```
addi $7,$4,0x7654
beq $7,$8,label1
nop
ori $1,$0,1
label1:
```