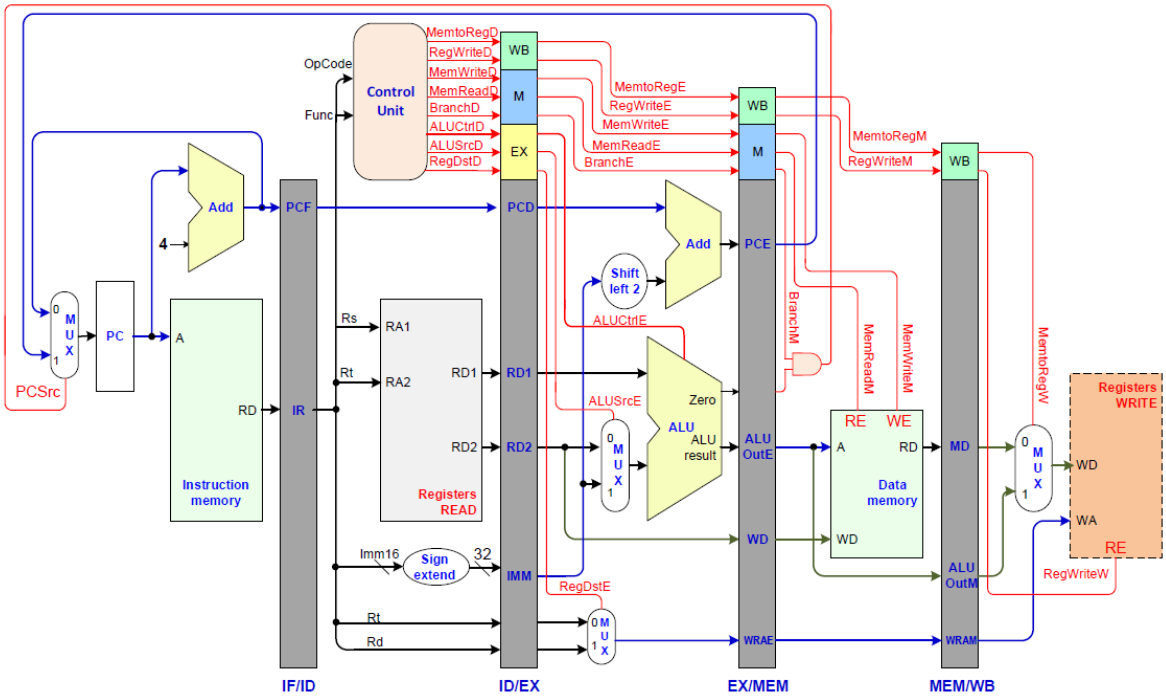
P6-Verilog流水线实验报告

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{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}。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

ALU操作信号

```
    assign ALU =      (ls_ins|add|addu|addi|addiu)?      4'd0://+
                      (sub|subu)?                        4'd1:  //-
                      (or0|ori)?                         4'd2:  //|
                      (and0|andi)?                       4'd3:  //&
                      (xor0|xori)?                       4'd4:  //^
                      (nor0)?                            4'd5:  //~|
                      (lui)?                             4'd6: //<<16
                      (sll|sllv)?                        4'd7:  //<<
                      (srl|srlv)?                        4'd8:  //>>
                      (sra|srav)?                        4'd9:  //>>>
                      (sltu|sltiu)?                      4'd10: //<u
                      (slt|slti)?                        4'd11: //<
                                                         4'd12;//null
```

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

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

multdiv

信号名	方向	位数	描述
clk	I	1	clk
reset	I	1	reset
OP	I	4	操作信号
A	I	32	操作数1
B	I	32	操作数2
HI	O	32	hi
LO	O	32	lo
busy	O	32	busy

ByteData

信号名	方向	位数	描述
ins	I	32	指令
data_old	I	32	原数据
WD	I	32	写入数据
data_new	O	32	新数据
addr	I	2	地址后两位

该模块用于sw，sh，sb指令的数据写入

ByteLoad

信号名	方向	位数	描述
ins	I	32	指令
RD	I	32	原数据
RD_new	I	32	新数据
addr	I	2	地址后两位

该模块用于load类指令的数据读出扩展

（四）重要机制实现方法

1. 跳转

ID级中判断是否跳转并把NPC传给IF

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

```
multu   $23,$23
nop
lh      $23,-224($23)
slt     $23,$23,$23
mflo    $1
mfhi    $2
addi    $2,$0,223
addi    $23,$0,166
mthi    $18
nop
lhu     $18,-160($18)
sltu    $30,$18,$18
mflo    $1
mfhi    $2
addi    $30,$0,40
mtlo    $0
nop
lw      $0,40($0)
sub     $16,$0,$0
mflo    $1
mfhi    $2
addi    $1,$0,19
addi    $16,$0,16
div     $24,$24
```

```
lhu     $18,-160($18)
sltu    $30,$18,$18
mflo    $1
mfhi    $2
addi    $30,$0,40
mtlo    $0
nop
lw      $0,40($0)
sub     $16,$0,$0
mflo    $1
mfhi    $2
addi    $1,$0,19
addi    $16,$0,16
div     $24,$24
nop
lb      $24,-144($24)
addiu   $24,$24,198
mflo    $1
mfhi    $2
addi    $2,$0,89
divu    $19,$19
```

三、思考题

- 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？

  乘除运算延迟较高，需要多个周期完成，ALU部分在一个周期完成计算

  与GRF寄存器分开设置，若直接存入GRF中的寄存器，容易发生冲突。而且乘除需要2个寄存器，GRF中寄存器位数不够

- 参照你对延迟槽的理解，试解释“乘除槽”。

  乘除运算时非乘除指令可照常向后流水

  可以通过编译，在乘除运算过程中填入不冲突HI和LO的指令，乘除计算和其他指令互不影响

- 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）

  字符串为顺序存储，按字节寻址可直接以此向后寻址，如果按照字访问需要将地址左移2位

- 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。

jal和使用$31寄存器指令的冲突。将PC+8在EX级并入ALUout往后传，利用ALUout的转发解决

```
jal label
add $31,$31,$31
```

乘除指令和其他指令的冲突。给乘除指令设置Tuse。

```
add $1,$1,$2
div $1,$1
mflo $1
```

冲突解决使用AT法，和P5完全一致，只是给新指令归类后加入控制器解析Tnew，Tuse

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

- 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？

采用指令分类，使代码更清晰。

```
     wire R_ins =
add|addu|sub|subu|slt|sltu|sll|srl|sra|sllv|srlv|srav|and0|or0|xor0|nor0;
     wire I_ins = addi|addiu|andi|ori|xori|lui|slti|sltiu;
     wire save_ins = sw|sh|sb;
     wire load_ins = lw|lh|lhu|lb|lbu;
     wire ls_ins = load_ins|save_ins;
...
    assign Tnew =   (load_ins)?2:
                    (R_ins|I_ins|jal|jalr)?1:0;
    assign rsTuse = (R_ins|I_ins|ls_ins)?1:0;
    assign rtTuse = (I_ins)?3:
                    (save_ins)?2:
                    (R_ins)?1:0;
```