Project6 Verilog开发流水线CPU

一、CPU设计方案综述

（一）总体设计概述

- 处理器为32位处理器，5级流水线设计
- 支持的指令集为`MIPS-C3={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}`
- 所有运算型指令均不考虑溢出带来的异常
- 需要考虑延迟槽
- 支持同步复位

（二）关键模块定义

1.IFU

- 介绍：介绍：内部包括PC（程序计数器）、IM（指令存储器）及相关逻辑，PC用寄存器实现，具有同步复位功能，复位值为起始地址，IM容量为16KB
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟控制信号
reset	I	1	同步复位信号
npc	I	1	下一个PC
WE	I	1	IFU使能信号
PC	O	32	当前指令地址
Instr	O	32	当前指令

- 功能定义

序号	功能名称	功能描述
1	同步复位	在时钟上升沿到来时，如果`reset`有效，则进行复位，复位PC位0x0000_3000
2	取指令	PC从IFU取出相应的指令
3	更新PC值	在时钟上升沿到来时，如果WE有效，则将PC更新为npc
4	暂停	当WE=0时，代表暂停操作

2.GRF

- 介绍：用具有写使能的寄存器实现，寄存器总数为32个，具有同步复位功能，其中0号寄存器始终为0
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟信号
reset	I	1	复位信号
A1	I	5	5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1
A2	I	5	5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2
A3	I	5	5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器
WE	I	1	写使能信号
WD	I	32	32位写入数据
PC	I	32	当前指令地址
RD1	O	32	A1所对应的寄存器的值
RD2	O	32	A2所对应的寄存器的值

- 功能定义：

序号	功能名称	功能定义
1	同步复位	在时钟上升沿到来时，如果reset有效时，则进行复位，将所有寄存器中存储的数值清零
2	读数据	读出A1、A2地址对应寄存器中所存储的数据到RD1、RD2
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中

3.DM

- 介绍：容量为4KB，具有同步复位功能
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
WE	I	1	写使能信号
Addr	I	32	写入地址
WD	I	32	写入的数据
PC	I	32	当前指令地址
RD	O	32	读出的数据

- 功能定义：

序号	功能名称	功能描述
1	复位	在时钟上升沿到来时，若reset有效，则进行复位
2	写入数据	当WE = 1时，将WD写入Addr地址中
3	读数据	RD始终为Addr地址的值

4.ALU

- 介绍：提供32位加、减、或运算以及大小比较功能，不检测溢出。
- 端口定义：

端口	方向	位宽	功能描述
SrcA	I	32	操作数1
SrcB	I	32	操作数2
ALUOP	I	3	ALU运算控制信号
Result	O	32	运算结果

- 功能定义：

序号	功能名称	功能描述
1	或运算	当ALUOP = 1，进行或运算
2	加法运算	当ALUOP = 2，进行加法运算
3	减法运算	当ALUOP = 3，进行减法运算
4	置高位操作	当ALUOP = 4，将B右移16位

5.Controller

- 介绍：产生控制信号
- 端口定义：

端口	方向	位宽	功能描述
I	I	32	当前指令
addu	O	1	addu信号
subu	O	1	subu信号
ori	O	1	ori信号
lw	O	1	lw信号
sw	O	1	sw信号
beq	O	1	beq信号
lui	O	1	lui信号
jal	O	1	jal信号
jr	O	1	jr信号
j	O	1	j信号
rs	O	5	当前指令的rs
rt	O	5	当前指令的rt
rd	O	5	当前指令的rd，当I=jal时，rd=31
R	O	1	除jalr、jr外的R型运算指令
load	O	1	需要从内存中取值的信号
store	O	1	需要存入内存的信号
type_i	O	1	I型运算指令
j_l	O	1	跳转并链接信号的值
j_ra	O	1	需要用到rs寄存器的跳转指令
j_2reg	O	1	需要用到双寄存器的跳转指令

- 功能定义：

序号	功能名称	功能定义
1	产生控制信号	产生控制信号

6.Ext

- 介绍：支持16bit到32bit的零拓展以及符号拓展
- 端口定义：

端口	方向	位宽	功能描述
imm_16	I	16	16位待拓展的立即数
ExtOp	I	1	拓展控制信号
Result	O	32	拓展后的结果

- 功能定义：

序号	功能名称	功能描述
1	零拓展	当ExtOp = 0时，进行零拓展
2	符号拓展	当ExtOp = 1时，进行符号拓展

7.IF_ID

- 介绍；I级和D级间的流水线寄存器，同时产生D级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
reset	I	1	复位信号
clk	I	1	时钟控制信号
F_I	I	32	IFU指令
F_PC	I	32	IFU的PC
D_I	O	32	D级指令
D_PC	O	32	D级PC
D_Branch	O	4	跳转信号

- 功能定义：

序号	功能名称	功能描述
1	流水F_I	在时钟上升沿到来时，将F_I赋给D_I
2	流水F_PC	在时钟上升沿到来时，将F_PC赋给D_PC
3	生成D级控制信号	生成D级控制信号

8.ID_EX

- 介绍：D级和E级间的流水线寄存器，同时产生E级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟控制信号
reset	I	1	复位信号
D_I	I	32	D级指令
D_PC	I	32	D级PC
D_RD1	I	32	D级RD1
D_RD2	I	32	D级RD2
E_I	O	32	E级指令
E_PC	O	32	E级PC
E_RD1	O	32	E级RD1
E_RD2	O	32	E级RD2
ExtOp	O	1	拓展信号
E_s	O	1	SrcB选择信号
E_ALUOP	O	1	ALU控制信号

- 功能定义：

序号	功能名称	功能描述
1	流水D_I	在时钟上升沿到来时，将D_I赋给E_I
2	流水D_PC	在时钟上升沿到来时，将D_PC赋给E_PC
3	流水D_RD1	在时钟上升沿到来时，将D_RD1赋给E_RD1
4	流水D_RD2	在时钟上升沿到来时，将D_RD2赋给E_RD2
5	产生E级控制信号	产生E级ExtOp、E_s、E_ALUOP信号

9.EX_MEM

- 介绍：E级和M级间的流水线寄存器，同时产生M级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟控制信号
reset	I	1	复位信号
E_I	I	32	E级指令
E_PC	I	32	E级PC
E_AO	I	32	E级AO
E_RD2	I	32	E级RD2
M_WE	O	1	M级写入信号
M_I	O	32	M级指令
M_PC	O	32	M级PC
M_AO	O	32	M级AO
M_RD2	O	32	M级RD2

- 功能定义：

序号	功能名称	功能描述
1	流水E_I	在时钟上升沿到来时，将E_I赋给M_I
2	流水E_PC	在时钟上升沿到来时，将E_PC赋给M_PC
3	流水E_AO	在时钟上升沿到来时，将E_AO赋给M_AO
4	流水E_RD2	在时钟上升沿到来时，将E_RD2赋给M_RD2
5	产生M级控制信号	产生M级控制信号

10.MEM_WB

- 介绍：M级和W级间的流水线寄存器，同时产生W级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
reset	I	1	复位信号
clk	I	1	时钟控制信号
M_I	I	32	M级指令
M_PC	I	32	M级PC
M_RD	I	32	M级RD
M_AO	I	32	M级AO
WE	O	1	GRF写入信号
ra	O	1	ra写入信号
RegDst	O	1	写入rd信号
GRF_PC	O	32	GRF的PC
GRF_I	O	32	GRF的I
W_RD	O	32	W级RD
W_AO	O	32	W级AO
WD_S	O	3	GRF写入数据选择信号

- 功能定义：

序号	功能名称	功能描述
1	流水M_I	在时钟上升沿到来时，将M_I赋给GRF_I
2	流水M_PC	在时钟上升沿到来时，将M_PC赋给GRF_PC
3	流水M_RD	在时钟上升沿到来时，将M_RD赋给W_RD
4	流水M_AO	在时钟上升沿到来时，将M_AO赋给W_AO
5	产生W级控制信号	产生W级控制信号

11.SFU

- 介绍：暂停 (Stall) 和 转发 (Forward) 的控制单元，产生两者的控制信号。
- 端口定义：

端口	方向	位宽	功能描述
d_I	I	32	D级指令
ex_I	I	32	E级指令
mem_I	I	32	M级指令
wb_I	I	32	W级指令
Stall	O	1	暂停信号
RD1_s	O	4	RD1选择信号 0：GRF[rs] 1：E级的PC + 8 2：M级的PC + 8 3：M级的AO
RD2_s	O	4	RD2选择信号 0：GRF[rs] 1：E级的PC + 8 2：M级的PC + 8 3：M级的AO
SrcA_s	O	4	SrcA选择信号 0：ID_EX寄存器的RD1 2：M级的PC + 8 3：M级的AO 4：W级的WD
SrcB_s	O	4	SrcB选择信号 0：ID_EX寄存器的RD2 2：M级的PC + 8 3：M级的AO 4：W级的WD
M_Data_s	O	4	M_Data选择信号 0：EX_MEM的AO 4：W级的WD

- 功能定义：

序号	功能名称	功能描述
1	D级RD1、RD2的选择信号	当D级比较器的两个输入与之前指令产生数据冲突时，进行转发
2	E级SrcA、SrcB的选择信号	当E级ALU的两个来源于之前指令产生数据冲突时，进行转发
3	M级Data的选择信号	当M级Data于之前指令产生数据冲突时，进行转发
4	暂停	当发生控制冲突时，IFU和IF_ID将暂停，同时清空ID_EX

15、MDU

- 乘除单元
- 端口定义：

端口	方向	位宽	功能描述
A	I	32	SrcA
B	I	32	SrcB
MDU_OP	I	32	乘除运算控制信号
start	I	1	乘除运算开始信号
clk	I	1	时钟信号
reset	I	1	复位信号
Busy	O	1	忙碌信号
hi	O	32	当前HI寄存器的值
lo	O	32	当前LO寄存器的值

- 功能定义：

序号	功能名称	功能描述
1	有符号乘	将`A * B`的运算结果放到`{HI,LO}`中，`A,B`被看作为有符号数
2	无符号乘	将`A * B`的运算结果放到`{HI,LO}`中，`A、B`被看作为无符号数
3	有符号除	将`A / B`的运算结果放到`{HI,LO}`中，`A、B`被看作为有符号数
4	无符号除	将`A / B`的运算结果放到`{HI,LO}`中，`A、B`被看作为无符号数
5	写`lo`寄存器	将A写入`lo`寄存器
6	写`hi`寄存器	将`A`写入`hi`寄存器

（三）重要机制实现方法

利用Tuse-Tnew法解决数据冒险

- 先分析每个指令的Tuse和Tnew，然后根据Tuse和Tnew产生暂停以及转发信号
- 如果Tuse<Tnew，则优先考虑转发，其次考虑暂停
- 暂停
  - `beq`需要用到`rs,rt`寄存器，所以对于任何**需要写入寄存器**的指令，都需要考虑暂停和转发，其中如果和`lw`发生数据冲突，需要暂停两个周期
  - `jr`需要用到`rs`寄存器，所以对于任何**需要写入寄存器**的指令，都要考虑暂停和转发，与`beq`的情况类似

    - 除跳转指令外**任何需要用到寄存器进行计算**的指令与`lw`指令之如果发生了数据冲突，需要暂停
- 转发
    - D级与W级的转发已通过GRF内部转发实现，故无需考虑
    - D级比较器的两个输入的转发来源
        - E级的PC + 8
        - M级的PC + 8
        - M级的AO
    - SrcA、SrcB的转发来源
        - M级的PC + 8
        - M级的AO
        - W级的WD
    - M_Data的转发来源
        - W级的WD

测试样例

```
ori $5, $0, 0x1234
ori $6, $0, 0xffff
sw $5, 0($0)
sh $6, 2($0)
sb $6, 4($0)
sb $6, 5($0)
sb $6, 6($0)
sb $6, 7($0)

ori $6, $0,0xf
sw $6, 0($0)
lhu $8, 0($0)
ori $6, $0,0xf0
sh $6, 2($0)
lhu $9, 2($0)
ori $6, $0, 0xf00
sb $6, 4($0)
lbu $10, 4($0)
ori $6, $0, 0xf000
sb $6, 5($0)
lbu $11, 5($0)
ori $6, $0, 56123
sb $6, 6($0)
lbu $12, 6($0)
ori $6, $0, 16521
sb $6, 7($0)
lbu $13, 7($0)
mult $5, $5
mfhi $7
ori $5, $0, 200
ori $6, $0, 30
mflo $8
mult $5, $6
mfhi $7
mflo $8
mult $6, $6
mfhi $7
```

```
mflo $8
ori $1, $0, 100
ori $2, $0, 200
ori $3, $0, 300
ori $4, $0, 400
ori $5, $0, 500
addu $6, $5, $1
addu $7, $5, $6
add $8, $6, $7
add $9, $8, $5
addi $10, $9, -100
addi $11, $10, 200
addiu $12, $10, 200
sw $12, 0($0)
addiu $13, $10, -11
and $14, $12, $13
and $15, $14, $13
andi $16, $15, 200
addi $16, $15, 0xffff
ori $1, $0, 0x000f
ori $2, $0, 0x00f0
ori $3, $0, 0x0f00
ori $4, $0, 0xf000
addu $5, $5, $4
nor $5, $5, $3
nor $6, $5, $4
sw $6, 0($0)
sw $5, 0($0)
ori $5, $0, 0xf000
sll $6, $5, 4
sll $7, $6, 4
sll $8, $7, 4
ori $5, $0, 0x8000
ori $6, $0, 5
sllv $7, $5, $6
sllv $8, $7, $6
sllv $8, $8, $8
sllv $9, $8, $7
ori $5, $0,250
sltiu $6, $5, 250
sltiu $6, $5, 240
sltiu $6, $5, 260
lui $5, 0x8000
slti $6, $5, 0
slti $6, $5, 0x200
slti $6, $5, 0xfff
lui $5, 0x8000
srl $6, $5, 5
sra $7, $5, 5
mult $6, $7
mflo $15
srlv $16, $15, $6
mtlo $16
mflo $15
sllv $17, $16, $15
bne $17, $16, jump
addi $5, $6, 500
sw $5, 1528($0)
```

```
jump:
sh $6, 1530($0)
lbu $25, 1530($0)
sb $25, 0($25)
lb $6, 0($25)
addi $8, $9, 215
xor $10, $9, $8
xor $11, $10, $8
```

二、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？
   - 首先乘除的运算需要的时钟周期数比普通的ALU运算要长，如果将乘除运算整合进ALU的话，那么CPU整体的时钟频率将降低，违背了设计流水线的初衷；反而如果我们将乘除法部件单独拿出来的话，那么其之后的与`HI,LO`无关的指令将可以继续执行，这样虽然没有减少乘除指令的周期数，但是却提高了CPU整体的时钟频率，无疑是一个非常好的设计。
   - 独立的`HI,lo`设立的目的方便乘除指令与其他指令并行执行。
2. 参照你对延迟槽的理解，试解释“乘除槽”。
   - 当乘除法进行或将开始时，乘除有关指令会被阻塞到`IF/ID`流水线寄存器，相当于处理乘除槽。
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）
   - 比如要存入一个字符串时，因为字符的长度只是8位的，所以一个字节就可以解决，这样的话按字节访存会比按字访存更加节约内存地址空间。比如一个7个字符的字符串，再加上最后的`\0`，总共8个字符。如果是按字节访存的话，则需8个字节，而如果是按字访存的话，则需要8个字，这样就浪费了很多空间，可见按字节访存可以更大限度地增大内存空间的利用率。
   - 其次就是当访问类型只占一个字节时，比如`char`。
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？

   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。
   - 冲突方面的话，结构冲突因为采用了DM和IM分离分策略所以不存在，那么主要就是看控制冒险和数据冒险，分别通过比较前移+延迟槽和暂停转发解决。
   - 构造策略：使用`ori`指令将每个寄存器赋值，然后将这些都存入DM，最后构造数据相关的运算指令，中间插入跳转指令，根据这样的规则生成了大量的测试样例，针对每种转发、暂停情况，都进行了相应的测试。
5. 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？
   - 采用了指令分类的思想，做P5时已经进行了一个初步的分类，P6指令的分类也很类似，所以转发策略不用进行改变，暂停部分只需考虑由于乘除指令延迟所造成的暂停即可。