P6 Verilog 流水线+CPU 设计文档

19373490-吴昊天

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 设计的流水线处理器，支持的指令为 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}。整体设计分为 5 级，对应 F、D、E、M、W，并设计转发和暂停单元以及乘除模块。顶层有效信号为时钟信号 clk 和复位信号 reset，复位为同步复位。

（二）关键模块定义

1、F 级

内部包括 PC（程序计数器）、IM（指令存储器）等相关逻辑。

（1）PC 用 32 位寄存器实现，具有同步复位功能，复位至 0x0000_3000.

表 1、PC 描述

功能描述	32 位可复位寄存器，reset 高电平有效。 同步复位至 0x0000_3000	
信号名	方向	描述
Clk	I	MIPS-C 处理器时钟
Reset	I	复位信号，同步复位
PCEn	I	PC 使能端
NPC[31:0]	I	32 位次地址输入
PC[32:0]	O	32 位输出，当前地址

（2）IM 用寄存器阵列实现，大小为 32bit×4096 字。初始读取指令时以系统任务$readmemh 读取 code.txt 的十六进制机器码。

表 2、IM 描述

功能描述	寄存器阵列，存储指令。以$readmemh 读取 code.txt 大小为 32×4096 字	
信号名	方向	描述
addr[31:0]	I	取值地址，取 addr[13:2]
Instr[31:0]	O	为读出的指令

2、D 级

（1）IF/ID 寄存器，为连接 F 级和 D 级的寄存器。

表 3、Dreg 描述

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
DregEn	I	D 级寄存器使能端
Instr_F[31:0]	I	F 级指令
PC_F[31:0]	I	F 级 PC
PC4F[31:0]	I	F 级 PC+4
Instr _D[31:0]	O	D 级指令
PC_D[31:0]	O	D 级 PC
PC4D[31:0]	O	D 级 PC+4

（2）GRF（通用寄存器组，也称为寄存器文件、寄存器堆）。具有写使能的寄存器阵列，大小为 32bit×32 字，具有同步复位功能。

表 4、GRF 描述

功能描述	寄存器组，reset 同步复位	
信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，同步复位
RFWr	I	使能信号，时钟上升沿时为 1 有效
A1[4:0]	I	第一个读出寄存器编号

A2[4:0]	I	第二个读出寄存器编号
A3[4:0]	I	回写寄存器编号
WData[31:0]	I	回写寄存器的数据值
WPC[31:0]	I	应题目输出要求，当前执行指令
RD1[31:0]	O	第一个读出的寄存器值
RD2[31:0]	O	第二个读出的寄存器值

（3）EXT 扩展

表 5、EXT 描述

信号名	方向	描述
Imm[15:0]	I	16 位立即数
EXTOp[1:0]	I	EXT 功能选择信号 00：符号扩展 01：无符号扩展 10：加载立即数至高位 11：保留
EXTout[31:0]	O	EXT 输出数据

（4）NPC 相当于一个组合逻辑，输入 PC 以及相关辅助信号，根据控制信号 NPCOp 选择计算方式，输出次地址 NPC。

表 6、NPC 描述

信号名	方向	描述
PC[31:0]	I	当前地址
Instr[31:0]	I	32 位指令
NPCOp[2:0]	I	NPC 地址选择： 000：PC+4（计算顺序地址） 001：计算 branch 地址 010：计算 j&jal 地址 011：计算 jr 地址

zero	I	rs 和 rt 相等判断 1：相等 0：不相等
RA[31:0]	I	32 位输入地址
NPC[31:0]	O	32 位次地址

3、E 级

（1）ID/IE 寄存器，为连接 D 级和 E 级的寄存器。

表 7、Ereg 描述

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
EClr	I	暂停清楚信号
Instr_D[31:0]	I	D 级流水指令
PC_D[31:0]	I	D 级流水 PC
A3D	I	D 级流水回写地址
RD1D[31:0]	I	D 级寄存器读出值 1
RD2D[31:0]	I	D 级寄存器读出值 2
EXToutD[31:0]	I	EXT 输出 32 位结果
Instr_E[31:0]	O	E 级指令
PC_E[31:0]	O	E 级 PC
A3E	O	E 级回写地址
RD1E[31:0]	O	E 级输入寄存器结果 1
RD2E[31:0]	O	E 级输入寄存器结果 2
EXToutE[31:0]	O	E 级输入 EXT 结果

（2）ALU 计算单元

ALU（算术逻辑单元）。提供 32 位加、减、或运算及比较大小功能，控制信号为 ALUOp。

表 8、ALU 描述

信号名	方向	描述
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
ALUOp[3:0]	I	ALU 功能选择 0000：A+B　0001：A-B 0010：A|B　0011：A&B 0100：A^B　0101：~（A|B） 0110：SLT　0111：SLTU 1000：SLL　1001：SRL 1010：SRA 其余保留
ALUout[31:0]	O	32 位计算结果

（3）乘除单元 MD_Unit

处理乘除及其相关指令的模块，其中分别模拟 5 个和 10 个乘除的时间延迟，通过 busy 位表示是否正在运算。

表 9、乘除模块描述

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
D1[31:0]	I	第一个操作数
D2[31:0]	I	第二个操作数
Mdop[2:0]	I	控制信号
start	O	开始运算
busy	O	正在运算
HIE[31:0]	O	读出 HI 数值
LOE[31:0]	O	读出 LO 数值

4、M 级

（1）IE/IM 寄存器，为连接 E 级和 M 级的寄存器

表 10、Mreg 描述

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
Instr_E[31:0]	I	E 级流水指令
PC_E[31:0]	I	E 级流水 PC
A3E	I	E 级流水回写地址
RD2E[31:0]	I	E 级输入值 2
ALUoutE[31:0]	I	E 级 ALU 计算结果
Instr_M[31:0]	O	M 级指令
PC_M[31:0]	O	M 级 PC
A3M	O	M 级回写地址
ALUout_M[31:0]	O	M 级 ALU 计算结果 （DM 写入地址）
RD2M[31:0]	O	M 级输入值 2 （DM 写入数据）

（2）DM（数据存储器）由寄存器阵列表示。Reset 同步复位，写使能 DMWr 为 1 时可以写入。

表 11、DM 描述

功能描述	寄存器阵列，存储数据。Reset 同步复位 以$display 按要求输出，大小为 32×4096 字	
信号名	方向	描述
Clk	I	时钟信号
Reset	I	同步复位信号
addr[31:0]	I	访问地址，取 addr[13:2]
WPC[31:0]	I	按要求输出当前 PC 地址
DMWr	I	DM 写使能信号
MemWrite[31:0]	I	DM 写入数据
MemRead[31:0]	I	DM 读出数据

5、W 级

（1）IM/IW 级寄存器，为连接 M 级和 W 级的寄存器。

表 12、Wreg 描述

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
Instr_M[31:0]	I	M 级流水指令
PC_M[31:0]	I	M 级流水 PC
A3M	I	M 级流水回写地址
ALUout_M[31:0]	I	M 级流水 ALU 计算结果
DMreadM[31:0]	I	M 级 DM 读出数据
Instr_W[31:0]	O	W 级指令
PC_W[31:0]	O	W 级 PC
A3W	O	W 级回写地址
ALUout_W[31:0]	O	W 级 ALU 计算结果
DMreadW[31:0]	O	W 级 DM 读出数据

（2）Load 模块

用于处理从 DM 中读出信号，以应对 lw,lhlhu,lb,lbu 的不同要求。

表 13、load 模块描述

信号名	方向	描述
DMreadW[31:0]	I	输入原始数据值
addr[31:0]	I	数据地址
LDop[2:0]	I	控制信号 000：lw 001：lb　010：lbu 011：lh　100：lhu
LoadData[31:0]	O	输出 load 数值

三、控制信号单元

本 CPU 中控制信号采用分布式译码，将执行指令流水，在需要时再分别译码，译码逻辑和单周期译码逻辑完全相同。

鉴于指令较多，本处理器采取现将同类指令分类，再译出控制信号的方式以降低复杂度。将指令分为 9 类，分别为 cal_r，shif，shifv，cal_i，load，store，branch，jump，md 等。

cal_r：add，addu，sub，subu，sll，srl，sra，sllv，srlv，srav，and，or，xor，nor，slt，sltu.

shif：sll，srl，sra.

shifv：sllv，srlv，srav.

cal_i：addi，addiu，andi，ori，xori，lui，slti，sltiu.

load：lw，lb，lbu，lh，lhu.

store：sw，sh，sb.

branch：beq，bne，blez，bgtz，bltz，bgez.

jump：j，jar，jalr，jr.

md：mult，multu，div，divu.

共有 8 个控制信号如下表：

信号名	描述
NPCOp[2:0]	NPC 地址选择： 000：PC+4（计算顺序地址） 001：计算 beq 地址 010：计算 jal 地址 011：计算 jr 地址
M1Sel[1:0]	回写寄存器选择：（写哪里） 00：回写 rt 01：回写 rd 10：回写$31 11：5’b0 不回写
M2Sel[1:0]	回写数据选择：（写什么）

	00：ALU 计算所得数据 01：DM 读出数据 10：PC+4 11：保留
RFWr	寄存器写使能信号 1：上升沿写入数据 0：写入无效
EXTOp[1:0]	扩展单元功能选择： 00：符号扩展 01：无符号扩展 10：加载立即数至高位 11：保留
M3Sel	ALU 计算数选择： 1: 寄存器读出数据二 0：扩展单元所得立即数
ALUOp[2:0]	ALU 功能选择 0000：A+B 0001：A-B 0010：A|B 0011：A&B 0100：A^B 0101：~（A|B） 0110：SLT 0111：SLTU 1000：SLL 1001：SRL 1010：SRA 其余保留
DMWr	DM 写使能信号： 1：上升沿数据写入 0：写入无效

指令\信号名	NPCOp	M1Sel	M2Sel	RFWr	EXTOp	M3Sel	ALUOp	DMWr

cal_r	000	01	00	1	00	0		0
shif	000	01	00	1	00	0		0
shifv	000	01	00	1	01	0		0
cal_i	000	00	00	1	*	1		0
load	000	00	01	1	00	1		0
store	000	11	00	0	00	1		1
branch	001	11	00	0	00	0		0
j	010	11	00	0	00	0		0
jal	010	10	10	1	00	0		0
jalr	011	01	10	1	00	0		0
jr	011	11	00	0	00	0		0

四、转发和暂停单元

转发和暂停分别用 ForwardUnit 和 StallUnit 来处理，分布式译码在每一级译出当前级需求者的读取地址以及供给者的回写地址，还有 Tuse 和 Tnew 等信息，输入到两个单元中处理是否转发及暂停。

	Tuse_D	
指令	rs	rt
cal_r	1	1
shif	3	1
shifv	1	1
cal_i	1	3
load	1	3
store	1	2
branch	0	0
j	3	3
jal	3	3
jalr	0	3

jr	0	3
md	1	1
mthi	1	3
mtlo	1	3

指令	Tnew_E
cal_r	1
shif	1
shifv	1
cal_i	1
load	2
store	0
branch	0
j	0
jal	1
jalr	1
jr	0
mfhi	1
mflo	1

指令	Tnew_M
cal_r	0
shif	0
shifv	0
cal_i	0
load	1
store	0
branch	0
j	0

jal	0
jalr	0
mfhi	0
mflo	0

四、测试样例阐述

说明文档附件含 test1-6，其中 test1-2 测试除乘除以外指令例如 cal_r，cal_i，load，store 等之间的冲突。

test3-4 测试 branch 以及以上指令的冲突。

test5-6 添加乘除指令，测试包括乘除指令与非乘除指令，以及乘除指令之间暂停是否正确，乘除槽是否正确。

五、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？

因为现实的处理器计算乘除法需要的周期数比 ALU 中其他计算的延迟要长，如果整合进 ALU 的话，在 ALU 这个部件就会成为关键路径延迟整个 CPU 的执行周期数。

用独立的 HI、LO 寄存器来保存乘除法的计算结果，乘除指令的计算没有在一个周期内结束，不能直接用流水线的寄存器来保存结果，此外如果在 GRF 中回写 HI/LO，就要一次写两个寄存器，很不方便。

2、参照你对延迟槽的理解，试解释“乘除槽”。

乘除槽是指在执行乘除指令时，其他非乘除指令也可以正常在流水线中执行，而后续的乘除相关指令则暂停，这样就避免因乘除指令而拖累整个 CPU 的性能。

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）

当执行的指令中 sb、sh、lb、lh 比较多的时候，按字节访存性能更优。

4、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？

首先将同类指令进行分类总和，分为 cal_r，shif，shifv，cal_i，load，store，branch，jump 以及 md 等，对一类指令统一处理，大大减少对于某条指令处理遗漏或赘余的可能性。

然后采用分布式译码，每一级单独定义一个 controller，控制信号用的时候再译码，减轻流水信号的压力。

尽量简化数据通路，比如 ALU 计算结果，PC+8 以及 MD_Unit 读出结果都走同一条路，转发时比较方便，且不用单独流水数据。

单独定义 load 模块以及对 DM 模块进行改造，以一次性添加 store 和 load 型指令，转发和暂停先考虑无乘除指令的处理，然后再考虑到乘除相关指令。

5、在线测试相关说明

首先进行功能测试，每两条指令之间插入 4 个 nop，检验基本部件和无转发数据通路有无问题。

再测试除 mult/div 外的指令是否正确，冲突测试与 P5 相同。

然后再增加 mult/div，先测试 mult/div 和其他指令冲突是否正确，再验证 mult/div 后跟乘除类指令是否正确的暂停。

再根据 Tuse 和 Tnew 的表格，反向构造样例。