Lab6 简单的类 MIPS 多周期流水化处理器实现

实验目的

- 理解CPU Pipeline、流水线冒险(hazard)及相关性，在lab5基础上设计简单流水线CPU
- 在1.的基础上设计支持Stall的流水线CPU。通过检测竞争并插入停顿（Stall）机制解决数据冒险/竞争、控制冒险和结构冒险
- 在2.的基础上，增加Forwarding 机制解决数据竞争，减少因数据竞争带来的流水线停顿延时，提高流水线处理器性能
  PS：也允许考虑将Stall与Forwarding结合起来实现
- 在3. 的基础上，通过predict-not-taken 或延时转移策略解决控制冒险/竞争，减少控制竞争带来的流水线停顿延时，进一步提高处理器性能

  PS：也允许考虑将2.、3.和4.合并起来设计
- 在4.的基础上，将CPU 支持的指令数量从9条或16条扩充为31条，使处理器功能更加丰富（选做）
- 在5.的基础上，增加协处理器CP0，支持中断相关机制及中断指令（选做）
- 应用 Cache 原理，设计 Cache Line 并进行仿真验证（选做）

实验原理

各模块原理分析

主控制单元模块

- 本次实验用到的控制信号有：
  - RegDst：目标寄存器的选择信号。（0：写入rt代表的寄存器，1：写入rd代表的寄存器）
  - ALUSrc：ALU的第二个操作数的来源。（0：使用rt寄存器中的数，1：使用立即数）
  - MemToReg：写寄存器的数据来源。（0：使用ALU的结果，1：使用从内存读取的数据）
  - RegWrite：写寄存器使能信号。（高电平表示当前指令需要写寄存器）
  - MemRead：读内存使能信号。（高电平表示当前指令需要读内存，比如load）
  - MemWrite：写内存使能信号。（高电平表示当前指令需要写内存，比如store）
  - beqSign：beq指令信号，高电平说明当前指令是beq指令。
  - bneSign：bne指令信号，高电平说明当前指令是bne指令。
  - extSign：符号扩展信号，高电平说明当前指令需要对操作数进行符号扩展。
  - JalSign：jal指令信号，高电平说明当前指令是jal指令。
  - JrSign：jr指令信号，高电平说明当前指令是jr指令。
  - luiSign：lui指令信号，高电平说明当前指令是lui指令。
  - ALUOp：发送给ALU控制器，用来进一步解析运算类型的控制信号。
  - Jump：无条件跳转信号。（高电平表示当前指令是无条件跳转指令，比如jump）
- 本次实验新增的指令操作码如下：

指令	opCode
bne	000101
addiu	001001
xori	001110
lui	001111
	001010
sltiu	001011

- 操作码对应的控制信号：

指令	opCode	regDst	aluSrc	regWrite	memToReg	memRead	memWrite	beqSign	bneSign	luiSign	extSign	jalSign	jrSign	aluOp	jump
R-Type	000000	1	0	1	0	0	0	0	0	0	0	0	0	1101	0
addi	001000	0	1	1	0	0	0	0	0	0	1	0	0	0000	0
addiu	001001	0	1	1	0	0	0	0	0	0	0	0	0	0001	0
andi	001100	0	1	1	0	0	0	0	0	0	0	0	0	0100	0
ori	001101	0	1	1	0	0	0	0	0	0	0	0	0	0101	0
xori	001110	0	1	1	0	0	0	0	0	0	0	0	0	0110	0
lui	001111	0	1	1	0	0	0	0	0	0	0	0	0	1010	0
lw	100011	0	1	1	1	1	0	0	0	0	1	0	0	0001	0
sw	101011	0	1	0	0	0	0	0	0	0	1	0	0	0001	0
beq	000100	0	0	0	0	0	0	1	0	0	0	0	0	0011	0
bne	000101	0	0	0	0	0	0	0	1	0	0	0	0	0011	0
slti	001010	0	1	1	0	0	0	0	0	0	1	0	0	1000	0
sltiu	001011	0	1	1	0	0	0	0	0	0	0	0	0	1001	0
jump	000010	0	0	0	0	0	0	0	0	0	0	0	0	1111	0
jal	000011	0	0	0	0	0	0	0	0	0	0	1	0	1111	1
jr	000000	0	0	0	0	0	0	0	0	0	0	0	1	1111	0

- 当出现不属于以上情况的操作码时，将所有控制信号置为0，视作空指令（nop）。

ALU控制单元模块

- 重新将 ALUOp 与 ALUCtr 进行联合设计。
    - ALUOp 将可以解析的指令的 ALUCtrOut 控制信号都解析完成，其余不能确定具体类型的 R 型指令留给运算单元控制器 (ALUCtr) 做进一步的解析，已经解析完成的指令的 ALUOp 直接可以成为运算单元控制器中ALUCtrOut 的输出。

指令	ALUOp/ALUCtrOut	ALU实际动作
addi	0000	带溢出检查的加法
addu, addiu, lw, sw	0001	不带溢出检查的加法
subi	0010	带溢出检查的减法
subu, beq, bne	0011	不带溢出检查的减法
and, andi	0100	逻辑与
or, ori	0101	逻辑或
xor, xori	0110	逻辑异或
nor	0111	伙计或非
slt, slti	1000	带符号小于时置位
sltu, slti	1001	无符号小于时置位
sll, sllv, lui	1010	逻辑左移
srl, srlv	1011	逻辑右移
sra, srav	1100	算术右移
j, jal, jr	1111	不做操作
R-type	1101	待解析

- 指令进一步解析时，还要产生一个控制信号：
    - ShamtSign：shamt选择信号，高电平说明操作数需从指令的shamt区域选取。
- ALU控制单元的进一步解析如下：

指令	Funct	Operation Control
add	100000	0000
addu	100001	0001
sub	100010	0010
subu	100011	0011
and	100100	0100
or	100101	0101
xor	100110	0110
nor	100111	0111
slt	101010	1000
sltu	101011	1001
sll	000000	1010，shamtSign = 1
srl	000010	1011，shamtSign = 1
sra	000011	1100，shamtSign = 1
sllv	000100	1010
srlv	000110	1011
srav	000111	1100
jr	001000	1111

ALU模块

- 输入与操作的对应关系在前一个模块的真值表中已反映。

寄存器组模块

- 与Lab4基本相同。
- 因为增加了jal指令，jal默认将下一条指令的地址保存至寄存器31，所以需要增加一个信号输入来表明这种情况。

数据存储器模块

- 与Lab4的存储器相同。

指令存储器模块

- 与Lab5相同。

有符号扩展单元模块

- 与Lab5相同。

程序计数器模块

- 不设置单独的PC模块。

- 利用阻塞赋值完成延迟到时钟上升沿赋值的操作。
- 简化为了一个顶层模块中的全局寄存器。

数据选择器模块

- 与Lab5相同。

流水线各阶段原理分析

取指（IF）

- 包含程序计数器和指令存储器模块。
- 该阶段根据程序计数器提供的PC从指令存储器中取出指令。

译码（ID）

- 包含主控制单元模块，寄存器组模块，有符号扩展单元模块，一个通过regDst控制信号在rt和rd之间选择运算结果的选择器。
- 该阶段对指令进行译码得到控制信号，进行有符号拓展与运算结果寄存器的选择。

执行（EX）

- 包含ALU控制单元，ALU以及一些对ALU操作数来源的选择器。
- 该阶段对指令进行进一步译码，以告知ALU应该执行什么运算动作。

访存（MA）

- 包含数据存储器，一个通过memToReg信号在ALU运算结果和访存结果之间选择执行结果的选择器。
- 该阶段进行内存的访问。

写回（WB）

- 该阶段进行对寄存器组的写回。
- 写回被安排在时钟下降沿进行操作，这样可以有效解决一部分的数据冒险。

流水线段寄存器

- 在流水线的两个阶段之间，设置一系列段寄存器，用于保存指令经过上一阶段执行后的结果。
    - IF -> ID 段：保存当前指令及其PC。
    - ID -> EX：保存控制信号（ALUOp、ALUSrc、luiSign、beqSign、bneSign、memWrite、memRead、memToReg、regWrite），有符号扩展单元的结果，解析出的指令中rs、rt、funct、shamt的值，读rs、rt寄存器得到的结果，当前指令的PC，目标寄存器的地址。
    - EX -> MA：保存控制信号（memWrite、memRead、memToReg、regWrite），ALU的运算结果，rt寄存器的值，目标寄存器的地址。
    - MA -> WB：保存regWrite控制信号，最终要写到寄存器的数据，目标寄存器的地址。

冒险解决机制原理分析

前向通路

- 添加两条前向通路：
    - 将读存储器的值直接送入下两条指令的EX阶段执行前进行选择。
    - 将EX阶段ALU的运算结果送入下一条指令的EX阶段执行前进行选择。
- 需要分别对rs与rt进行选择，故需要4个选择器，对应的选择信号为是否写/读寄存器和目标寄存器和目前寄存器是否相同。

插入停顿

- 有了前向通路的优化，现在只有在出现 load-use 情况的时候才需要流水线后续指令停顿一个周期。
- 用STALL信号来检测这种冒险。(根据相邻两条指令的相应控制信号判断)

预测不转移

- 条件转移：
  - 预测错误时，产生BRANCH信号，清除转移指令后的所有指令，接着转移到正确的地址继续执行。
  - 因为转移错误在EX阶段执行后发现，所以会造成流水线停顿两个周期。
- 非条件转移：
  - 将其提前至译码阶段处理。
  - 这样不会造成流水线停顿。

实现细节

- Ctr.v
  - 对比Lab5，添加了新的信号和新的操作码分支。
- ALUCtr.v
  - 对比Lab5，修改了ALUOp->ALUCtrOut的译码方式
- ALU.v
  - 对比Lab5，添加了新的运算类型分支。
- Registers.v
  - 与Lab5基本一致。
  - 添加了一个供jal指令使用的信号。
- dataMemory.v
  - 与Lab5基本一致。
- SignExt.v
  - 与Lab5基本一致。
- instMemory.v
  - 与Lab5基本一致。
- Mux32.v
  - 与Lab5基本一致。
- Mux5.v
  - 与Lab5基本一致。
- Top.v
  - 按照流水线各个阶段顺序实例化模块对象，并创建需要用到的总线。
  - 在每两个阶段之间创建段寄存器。
  - 为每个冒险解决机制创建总线和选择器。
  - always块每周期触发一次，采用时序逻辑赋值的方式连接各个阶段，实现指令流水。
- mem_data.dat
  - 在内存中放入了如下数据：

地址	数据	地址	数据	地址	数据	地址	数据
1	000000FF	9	00000004	17	00000008	25	000002FF
2	00000100	10	00000005	18	00000009	26	000003FF
3	00000101	11	00000006	19	0000000A	27	000004FF
4	00000102	12	00000007	20	00000107	28	000005FF
5	00000000	13	00000103	21	00000108	29	000006FF
6	00000001	14	00000104	22	00000109	30	000007FF
7	00000002	15	00000105	23	0000010A	31	000008FF
8	00000003	16	00000106	24	000001FF	32	000009FF

- mem_inst.dat
    - 用下列指令进行测试：

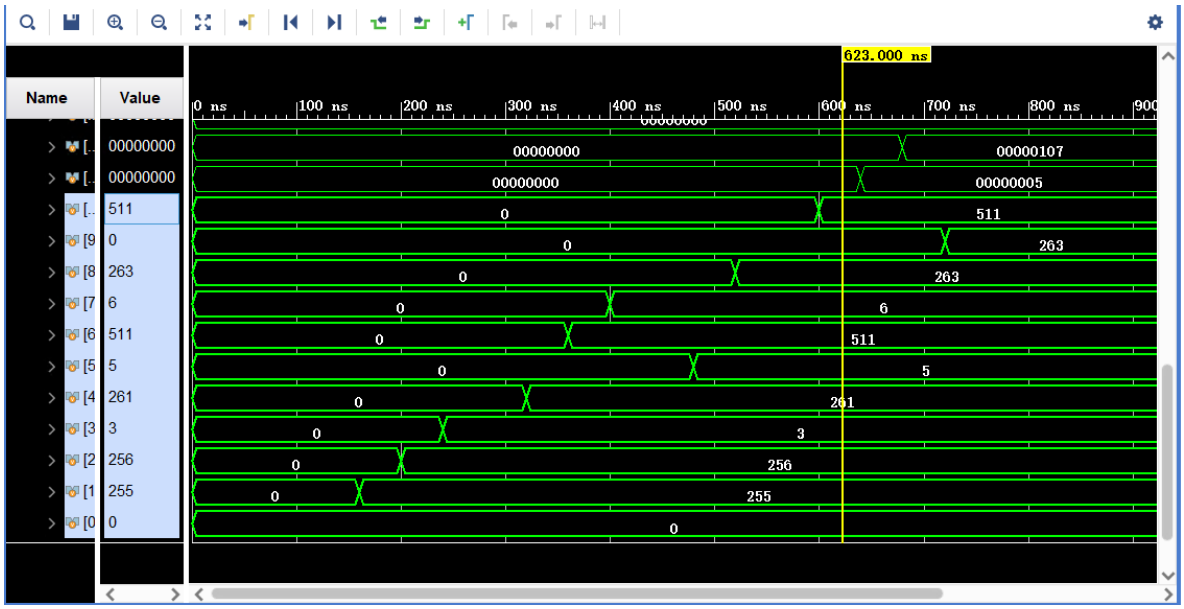
指令地址	指令	执行结果
0	lw $1, 0($0)	$1 = Mem[0] = 255
1	lw $2, 1($0)	$2 = Mem[1] = 256
2	lw $3, 7($0)	$3 = Mem[7] = 3
3	lw $4, 11($3)	$4 = Mem[14] = 261
4	add $6, $1, $2	$6 = 255 + 256 = 511
5	sub $7, $4, $1	$7 = 261 - 255 = 7
6	lw $16, 5($7)	$16 = Mem[11] = 7
7	and $5, $1, $4	$5 = 255 & 261 = 5
8	or $8, $16, $2	$8 = 7 | 256 = 263
9	sw $8, 7($5)	Mem[12] = 263
10	addi $10, $1, 256	$10 = 255 + 256 = 511
11	addi $11, $4, 255	$11 = 261 & 255 = 5
12	ori $12, $16, 256	$12 = 7 | 256 = 263
13	lw $9, 12($0)	$9 = Mem[12] = 263
14	beq $9, $8, 1	go to 16
15	sll $13, $11, 4	not executed
16	sll $14, $11, 4	$14 = $11 << 4 = 80
17	sll $15, $10, 4	$15 = $10 >> 4 = 31
18	slt $17, $15, $14	$17 = 1
19	slt $18, $15, $16	$18 = 0
20	j 22	go to 22
21	slt $18, $15, $14	not executed
22	jal 24	go to 24
23	beq $16, $11, 3	go to 27
24	addi $11, $11, 2	$11 = 5 + 2 = 7
25	jr $31	go to 23
26	addi $11, $11, 2	not executed
27	addi $11, $11, 2	$11 = 7 + 2 = 9

- Top_tb.v
    - 创建Top模块的实例化对象processor。
    - 从外部文件读入指令数据和内存数据。

- 生成周期为40ns的时钟信号，作为激励信号。

实验结果

- 通过在console输出信息，可以得到部分指令执行的结果，以此检查设计是否正确。
- 仿真结果：



总结与反思

- 复习了MIPS多周期处理器的指令流水线，指令流水过程中可能发生的冒险和几种解决机制，对各个指令的执行过程有了更加深刻的理解。
- 感谢学长学姐以及网络上的一些陌生人提供开源的学习资料、电路设计图、测试样例。
- 在网上找到了很多其他人设计的MIPS处理器，发现对于控制信号、一些译码细节上有所不同，仔细分析下来发现都是正确的，这说明只要是符合原理的设计，都可以实现这个处理器，而且不同的设计方式有不同的优点。比如在本次实验中将ALUOp和ALUCtrOut联合译码，可以简化一些译码步骤。还有一些其他巧妙的设计，可能在本次实验中没有采用，但是确实学习到了，可能会在以后的一些场合发挥作用。