RISC-Z2 处理器基础任务实验报告(任务 1-3)

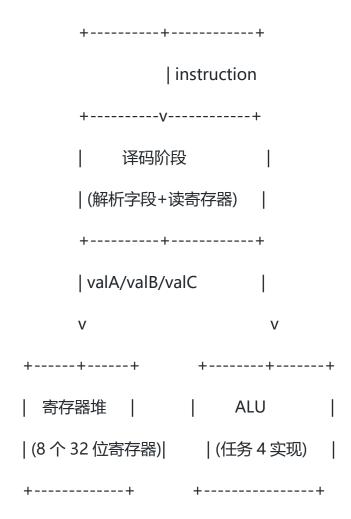
一. 设计文档

(一) 处理器整体设计思路

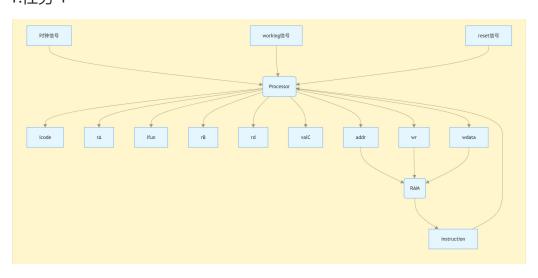
- 1.基础任务 1 设计思路: 创建 processor.v 和 ram.v 两个文件。ram.v 负责指令的存储与读取, processor.v 根据 working 信号决定执行取指或写入操作。取指时从 ram 读取指令并解析输出,写入时将数据写入 ram 指定地址。
- 2.基础任务 2 设计思路: 在基础任务 1 基础上,新增 regfile.v 实现寄存器堆功能。processor.v 在识别到 IRMOV 指令时,解析指令字段,将立即数零扩展后通过 regfile.v 写入目标寄存器,同时保持对存储器写入操作的支持。
- 3.基础任务 3 设计思路: 在基础任务 2 基础上,于 processor.v 中集成译码逻辑,从取指结果解析指令字段,并从 regfile.v 读取源寄存器值。修改 regfile.v 以支持双寄存器读取,同时在 processor.v 新增输出端口输出译码结果。

(二) 数据通路与控制通路框图

+-		+
	RAM	
	(存储指令和数据)	
+		+
	addr/data	
+-	V	+
	取指阶段	
I	(更新 PC,读取指令)	Ι



1.任务 1

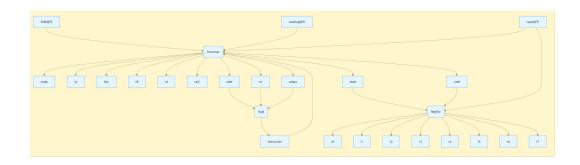


解释:

时钟信号和控制信号 (working、reset) 输入到 Processor 模块; Processor 模块通过 addr、wr、wdata 与 RAM 模块交互; RAM 将 instruction 输出给 Processor;

Processor 解析指令并输出各字段 (icode, rA, ifun, rB, rd, valC) 。

2.任务 2



解释:

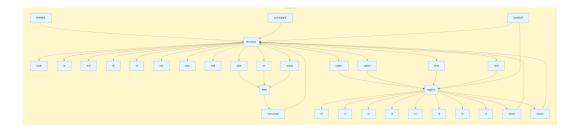
在基础任务 1 的基础上增加了 RegFile 模块;

Processor 通过 dstM 和 valM 控制 RegFile 的写入操作;

RegFile 输出 8 个寄存器的值 (r0-r7);

当识别到 IRMOV 指令时, Processor 将立即数通过 valM 写入指定寄存器。

3.任务3



解释:

在基础任务 2 的基础上增强了 RegFile 模块;

Processor 通过 rAddr1 和 rAddr2 指定要读取的寄存器;

RegFile 通过 rData1 和 rData2 返回读取的寄存器值;

Processor 将 rData1 和 rData2 分别输出为 valA 和 valB;

译码逻辑从 instruction 中解析出 rA 和 rB, 并用于读取寄存器值。

二. 验证材料

(一) testbench 代码说明

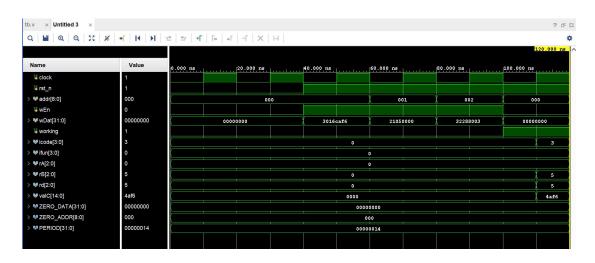
1,任务 1: 首先定义测试平台的信号,实例化 processor 模块。通过 initial 块生成 50MHz 时钟。在另一个 initial 块中,先初始化信号,然后在每个时钟周期 (20ns)向 ram 写入指令,最后使 working 信号有效,开启取指操作,并使用\$monitor 监控取指结果。

2.任务 2: 测试平台定义相关信号并实例化 processor 模块。生成 50MHz 时钟后,先复位寄存器堆,再依次写入 4 条 IRMOV 指令,最后使 working 信号有效执行指令,使用\$monitor 监控寄存器值变化。

3.任务 3: 同样定义信号并实例化 processor 模块, 生成时钟。先复位寄存器堆,接着写入 4 条指令,最后开启执行与译码,使用\$monitor 监控译码输出。

(二) vivado 仿真结果截图

1.任务 1:



波形说明:

working=0 时: 成功写入指令到地址 0-2;

working=1 后: PC 从 0→1→2 顺序变化;

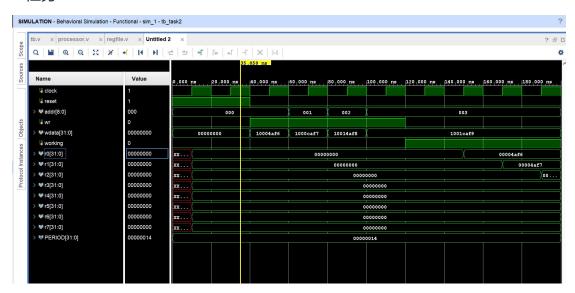
输出信号:

PC=0 时: icode=1 (IRMOV), rd=5, valC=12;

PC=1 时: icode=2 (ADD) , rA=2, rB=3;

在 working 信号置为 1 后,处理器开始取指,从波形图可看到不同时钟周期下 各指令字段的解析结果与预期相符,表明取指及更新 PC 操作功能正常。

2.任务 2:



波形说明:

复位后: r0-r7 全部为 0

执行阶段:

第 1 周期: r0 变为 10 (valC 零扩展)

第 2 周期: r1 变为 11

关键信号: dstM 正确匹配目标寄存器编号

复位信号有效时,寄存器堆清零。执行 IRMOV 指令后, r0 - r3 的值按预期更新, r4 - r7 保持为 0, 验证了 IRMOV 指令的写回操作和寄存器堆功能。

3.任务 3:

SIM	SIMULATION - Behavioral Simulation - Functional - sim_1 - tb															
m	ram.v x processor.v x regifile.v x tb.v x Untitled 15 x															
Scope	Q H Q Q X X	•[H H ·	± ± + [+	»Г] -	-F X I-I											
															280.000 ns	
Sources	Name	Value														
-	¼ clock	1	0.000 ns		0.000 ns		100.000	18	150.000 r	15	200	.000 ns		250.000 ns	300	
	¼ rst_n	1														
	> W addr[8:0] 000		000		001 002		003		000							
scts	¹ ⊌ wEn	0													<u> </u>	
Objects	> WwDat[31:0] 00000000		00000000	0850.	10a6	34a0	106e	Χ			0000000)				
Protocol Instances	> W r0[31:0]	00000000						00000000								
ul los	> V r1[31:0] 00000000							00000000								
rotoc	> № r2[31:0] 00000000 > № r3[31:0] 00000000		———					00000000								
а.	> ♥ r3[31:0] > ♥ r4[31:0]	———					00000000							≓		
	> V r5[31:0]	———					00000000							Ħ		
	> W r6[31:0]						00000000							ř l		
	> ₩r7[31:0]						00000000							Ĭ I		
	> W icode[3:0]				0				1	3) i	χ	0		j l	
	> W ifun[3:0]	0			0			8	0	4	Χ		0		Ď l	
	> W rA[2:0]	0			0			2	X	5	3)(0			
	> ♥ rB[2:0] 0				0			(4	1	(0	3	X	0			
	> V rd[2:0] 0				0				4		X 4		۰			
	> ♥ valC[14:0] 0000 > ♥ valA[31:0] 00000000				0000			000c	0000	0003			0000			
	> V valA[31:0]						00000000									
	> V valB[31:0] > V PERIOD[31:0]	00000000	· · · · · · · · · · · · · · · · · · ·					0000000								
	> W ZERO_DATA[31:0]	00000014	<u> </u>					00000014								
	> W ZERO_ADDR[8:0]	000	· · · · · · · · · · · · · · · · · · ·					000							=	
								- 000								

波形说明:

IRMOV 指令:

valA和 valB显示为 0 (未使用源寄存器)。

ADD 指令:

valA 显示 r2 的值(初始为 0);

valB 显示 r3 的值 (初始为 0);

关键验证点: rA/rB与 valA/valB的对应关系正确。

在执行与译码阶段,各指令的译码输出符合预期。如 IRMOV 指令的 valC 正确解析为立即数,ADD 指令正确读取源寄存器值到 valA 和 valB,证明译码操作功能正常。

三. 结果分析与解释

1.任务 1: 取指及更新 PC 操作成功实现。working 信号有效时,处理器按预期从 ram 读取指令并解析输出。这是因为 processor.v 中根据 working 信号选择从 ram 取指或向 ram 写入数据,且 PC 指针在每个时钟周期正确加 1,保证了指令的顺序读取。

2.任务 2:IRMOV 指令的写回操作正常。复位后寄存器堆清零,写入 IRMOV 指令执行后,目标寄存器的值更新为对应立即数。 这得益于 regfile.v 中写回逻辑的正确实现,以及 processor.v 对 IRMOV 指令的正确解析和对 regfile 的正确调用。

3.任务 3:译码操作准确完成指令字段解析和源寄存器值读取。从仿真结果看,不同指令的 icode、ifun、rA、rB、valC 等字段解析无误,且能从寄存器堆正确读取 valA 和 valB。这是由于 processor.v 中新增译码逻辑正确解析指令,同时 regfile.v 扩展后能支持双寄存器读取,二者协同工作实现了译码功能。