

实验 7-存储器与控制器实验

一、 实验准备

本次实验开始涉及 MIPS 架构 CPU 的设计, 其中涵盖 CPU 在流水线设计中所分割的两个阶段, 以下为实验概述: MIPS 架构 CPU 的传统流程可分为取指、译码、执行、访存、回写 (Instruction Fetch, Decode, Execution, Memory Request, Write Back) 五阶段。实验一完成了执行阶段的 ALU 部分, 并进行了简单的访存实验, 本实验将实现取指、译码两个阶段的功能。在进行本次实验前, 你需要具备以下实验环境及基础能力:

1. 了解 Xilinx Block Memory Generator IP 的使用
2. 了解数据通路、控制器的概念

二、 实验目的

- 2.1 了解随机存取存储器 RAM 的原理。
- 2.2 掌握调用 Xilinx 库 IP(Block Memory Generator)实例化 RAM 的方法;
- 2.3 掌握单周期 CPU 各个控制信号的作用和生成过程。
- 2.4 掌握单周期 CPU 控制器的工作原理及其设计方法。
- 2.5 理解单周期 CPU 执行指令的过程。
- 2.6 掌握取指、译码阶段数据通路、控制器的执行过程。

三、 实验设备

PC 机一台, Basys3 开发板, Xilinx Vivado 开发套件。

四、 实验任务

4.1 实验要求

图 1 为本次实验所需完成内容的原理图, 依据取指、译码阶段的需求, 分别需要实现以下模块:

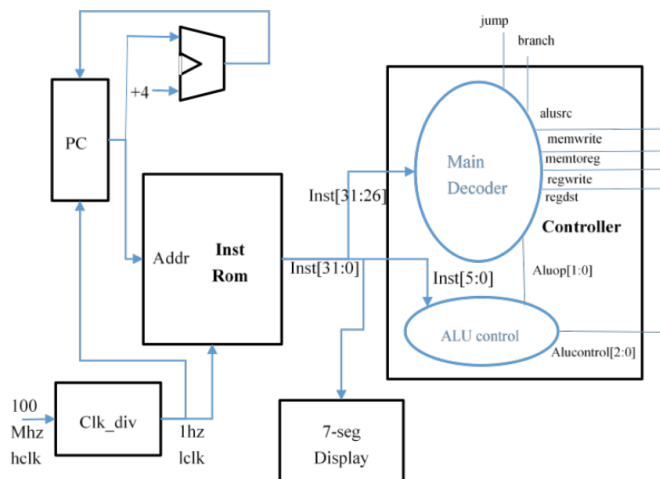


图 1

1. PC。D 触发器结构, 用于储存 PC(一个周期)。需实现 2 个输入, 分别为 clk, rst, 分别连接时钟和复位信号; 需实现 2 个输出, 分别为 pc, inst_ce, 分别连接指令存储器的 addra, ena 端口。其中 addra 位数依据 coe 文件中指令数定义;
2. 加法器。用于计算下一条指令地址, 需实现 2 个输入, 1 个输出, 输入值分别为当前指

令地址 PC、32'h4;

3. Controller。其中包含两部分:

(a). main_decoder。负责判断指令类型,并生成相应的控制信号。需实现 1 个输入,为指令 inst 的高 6 位 op,输出分为 2 部分,控制信号有多个,可作为多个输出,也作为一个多位输出,具体参照 4.3 进行设计;aluop 传输至 alu_decoder,使 alu_decoder 配合 inst 低 6 位 funct,进行 ALU 模块控制信号的译码。

(b). alu_decoder。负责 ALU 模块控制信号的译码。需实现 2 个输入,1 个输出,输入分别为 funct,aluop;输出位 alucontrol 信号。

(c). 除上述两个组件,需设计 controller 模块顶层文件调用两个 decoder,对应实现 op,funct 输入信号,并传入调用模块;对应实现控制信号及 alucontrol,并连接至调用模块相应端口。

4. 指令存储器。使用 Block Memory Generator IP 构造。(参考实验 5)

5. 时钟分频器。将板载 100Mhz 频率降低为 1hz,连接 PC、指令存储器时钟信号 clk。

注意: 板上只有 16 位的 segment 显示,所以只接入低 16 位进入显示(和实验 6 一样)

4.2 实验步骤

1. 从实验 5 实验中,导入 Display、clk_div 模块
2. 创建 PC 模块
3. 创建 main_decde, alu_decode 模块
4. 创建 Controller,调用 main_decode, alu_decode (逻辑参照表 2 和表 4)
5. 使用 Block Memory,导入 coe 文件(根据实验 6)
6. 自定义顶层文件,连接相关模块

Controller 输出信号与 led 管脚对应关系如下表:

memtoreg	memwrite	pcsrc	alusrc	regdst	regwrite	jump	branch	alucontrol
[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[2:0]
led[0]	led[1]	led[2]	led[3]	led[4]	led[5]	led[6]	led[7]	led[8:10]

表 1: 输出信号与 led 管脚

4.3 实验原理

4.4.1 取指阶段原理

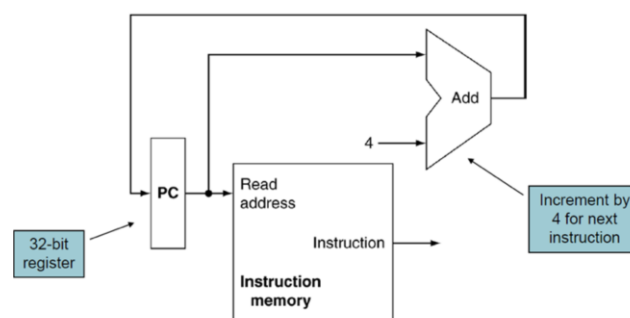


图 2

如图 2 所示, PC 为 32bit(1 word)的寄存器,其存放指令地址,每条指令执行完毕后增加 4,

即为下一条指令存放地址。指令地址传入指令存储器，即可取出相应地址存放的指令。需要注意的是，MIPS 架构中，采用字节读写，32bit word = 4 byte，故需要地址 +4 来获取下一条指令。

4.4.2 指令译码原理

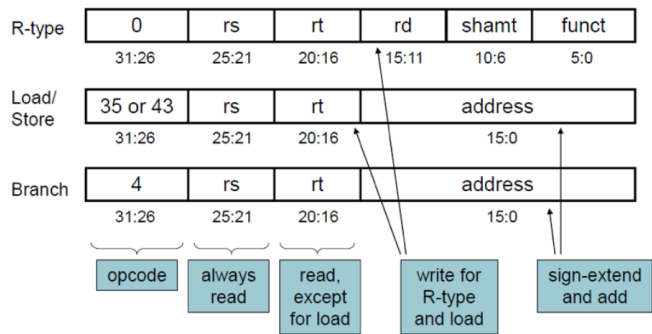


图 3

如图 3 所 32 位 MIPS 指令在不同类型指令中分别有不同结构。但[31:16]表示的 opcode，以及 [5:0] 表示的 funct，为译码阶段明确指令控制信号的主要字段。表 3 为 opcode 及 funct 识别得到的部分信号，详细信号表课堂 PPT。

表 2

opcode	aluop	operation	funct	alu function	alu control
lw	00	Load word	XXXXXX	Add	010
sw	00	Store word	XXXXXX	Add	010
beq	01	Branch equal	XXXXXX	Subtact	110
R-type	10	Add	100000	qdd	010
		subtract	100010	Subtract	110
		and	100100	And	000
		or	100101	Or	001
		set-on-less-than	101010	SLT	111

4.4.3 控制器实现原理

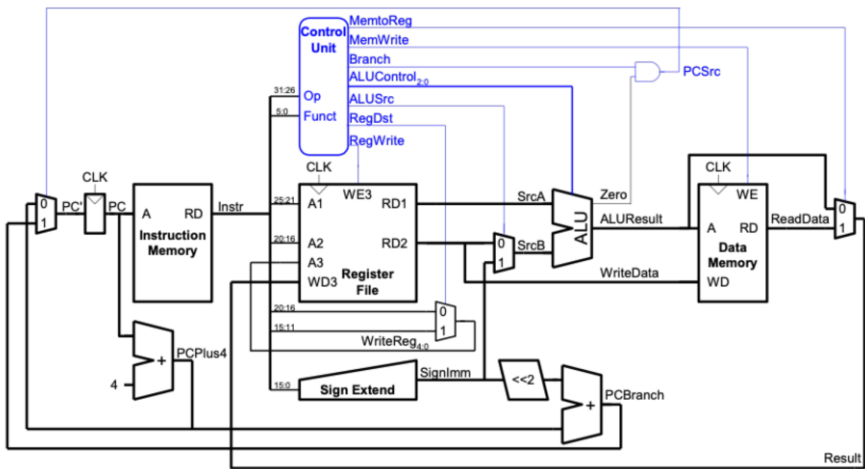


图 4

由图 4 可知，控制器输出的控制信号，用于控制器件的使能和多路选择器的选择，因此，根据不同指令的功能分析其所需要的路径，即可得到信号所对应的值。在此之前，参照下对各

个控制信号的含义进行理解。

表 3

信号	含义
memtoreg	回写的数据来自于 ALU 计算的结果/存储器读取的数据
memwrite	是否需要写数据存储器
pcsrc	下一个 PC 值是 PC+4/跳转的新地址
alusrc	送入 ALU B 端口的值是立即数的 32 位扩展/寄存器堆读取的值
regdst	写入寄存器堆的地址是 rt 还是 rd,0 为 rt,1 为 rd
regwrite	是否需要写寄存器堆
branch	是否为 branch 指令,且满足 branch 的条件
jump	是否为 jump 指令
alucontrol	ALU 控制信号,代表不同的运算类型

分析数据通路图，判断指令是否需要写寄存器、访存等等操作，以产生相应的控制信号。下面给出参考信号表：

表 4

instruction	op5:0	regwrite	regdst	alusrc	branch	memWrite	memtoReg	aluop1:0
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	001000	1	0	1	0	0	0	00
j	000010	0	X	X	X	0	X	XX

表四里面没有定于 pcsrc, 这里 pcsrc 只用于决定 beq 的 PC 选择，因此可以定义为：pcsrc = branch & zero。

4.4 实验报告要求

1. 按照实验要求 4.1 搭建如图 1 所示实验系统，需要在实验 6 的基础上，需要自己实现实现分频 clk_div.v, D 触发器结构 pc.v, 控制器 controller.v 模块，其中 controller.v 包含 main controller 和 ALU controller。
2. 在指令存储器里面依次放入 R-type (例如 add 和 sub), lw, sw, beq, ori 指令的 32 位机器码 (表 4)。根据读出的指令类型，根据 LED 灯去观察控制器的输出结果。
3. 在实验报告中，阐述实验设计过程，代码模块结构，以及观察到实验结果，并分析实验结构。