实验一报告

学号：2016K8009908007
姓名：薛峰

一、实验任务

实验目的：

（一）、熟悉实验平台；

（二）、将组成原理实验课的多周期 CPU 移植到本课程的实验平台；

（三）、熟练掌握 verilog 语言的基本用法，并规范 verilog 代码风格。

检验方法：

（一）、仿真检验时，控制台打印出”PASS”；

（二）、上板运行结果正确。

二、实验设计

（一）总体设计思路

该项目总体分为以下四个模块：mycpu_top，控制模块（cpu_control），寄存器堆(reg_file)，ALU。

其中 mycpu_top 模块为顶层模块，控制其他三个模块进行工作。各模块功能如下。

（二）mycpu_top 模块设计

1、工作原理

该模块是其余三个模块的顶层模块，用于控制状态机、读取内存的数据和指令、执行指令等功能。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号，来自 clk_pll 的输出时钟
resetn	IN	1	复位信号，低电平同步
inst_sram_en	OUT	1	ram 使能信号，高电平有效
inst_sram_wen	OUT	4	ram 字节写使能信号，高电平有效
inst_sram_addr	OUT	32	ram 读写地址，字节寻址
inst_sram_wdata	OUT	32	ram 写数据
inst_sram_rdata	IN	32	ram 读数据
data_sram_en	OUT	1	ram 使能信号，高电平有效
data_sram_wen	OUT	4	ram 字节写使能信号，高电平有效
data_sram_addr	OUT	32	ram 读写地址，字节寻址
data_sram_wdata	OUT	32	ram 写数据
data_sram_rdata	IN	32	ram 读数据
debug_wb_pc	OUT	32	写回级的 PC
debug_wb_rf_wen	OUT	4	写回级写寄存器堆(regfiles)的单字节写使能，若 mycpu 写 regfiles 为单字节写使能，则将写使能扩展成 4 位即可
debug_wb_rf_wnum	OUT	5	写回级写 regfiles 的目的寄存器号
debug_wb_rf_wdata	OUT	32	写回级写 regfiles 的写数据

3、结构设计图及功能描述

结构图如下(只包含部分指令，不含状态机)：

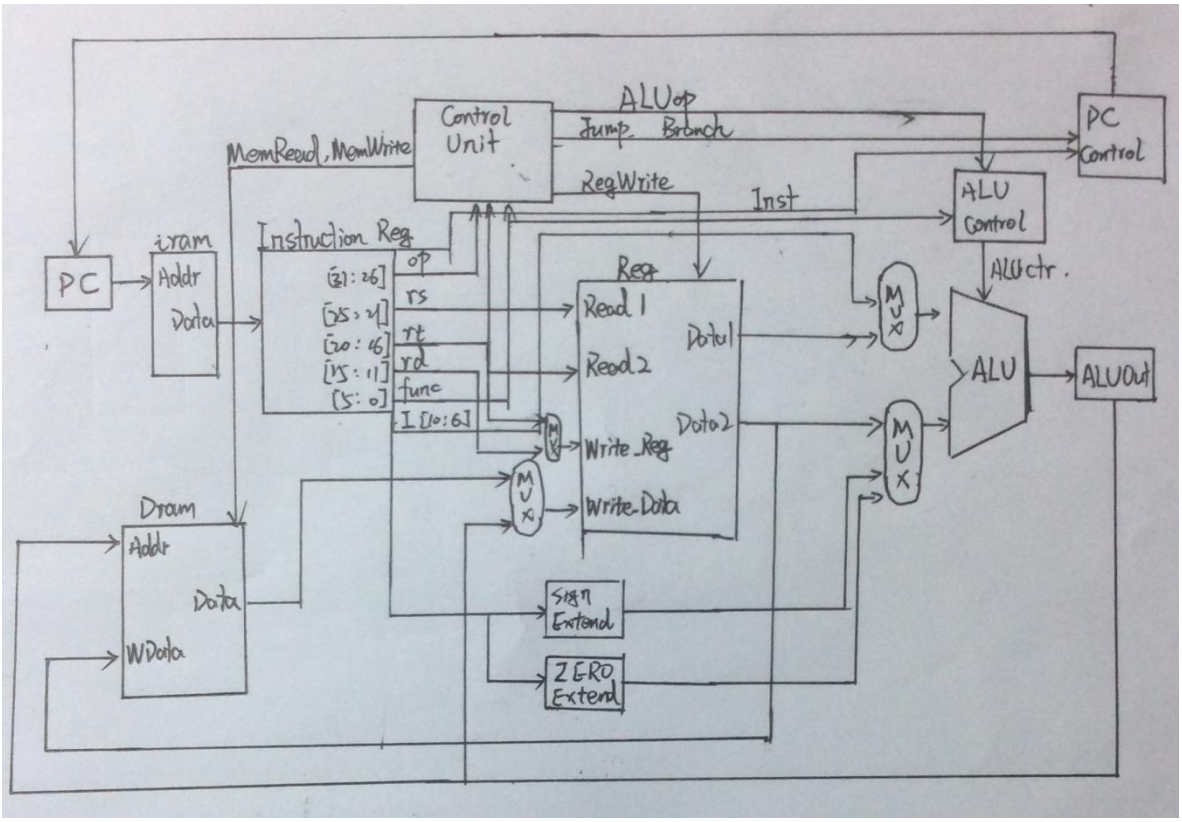


图 1：mycpu_top 结构图

首先，该模块实现了一个状态机，状态机包含 8 个状态，分别是取指(IF), 等待指令(IW)，译码(ID)，执行(EX)，内存写(ST)，内存读(LD)，读数据等待(RDW)，写回(WB)。 IF，IW，ID，ST，LD，RDW，WB 这七个状态之后的状态都是固定的，EX 状态之后的状态需要根据指令来判断。

其次，该模块将当前状态和指令输入至控制模块，根据控制模块产生的控制信号进行下一步操作。

该模块的主要部分如下：

对于 PC 的更新部分，需要根据控制信号 PCWrite 和 Jump 来进行 PC 的更新；

对于写寄存器的地址，需要根据控制信号 RegDst 来判断；

对于写寄存器的值，需要根据控制信号 MemtoReg 和 ALU 的结果 Result_op 来判断；

对于 ALU 的第一个输入值，根据指令判断，第二个输入值根据控制信号 ALUSrc 判断；

对于 ALU 的操作，根据控制信号 ALUOp 判断；

对于写内存的值，需要根据指令高 6 位和 Result_op 判断。

(三)cpu_control 模块设计

1、工作原理

该模根据 mycpu_top 传进来的信号输出控制信号。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号，来自 mycpu_top 的输出时钟
resetn	IN	1	复位信号，低电平同步
op	IN	6	指令的高 6 位，来自 mycpu_top 的 inst_sram_rdata[31:26]
rt	IN	5	指令的第 20 到 16 位，来自 mycpu_top 的 inst_sram_rdata[20:16]
func	IN	6	指令的第 6 位，来自 mycpu_top 的 inst_sram_rdata[5:0]
ea	IN	2	来自 mycpu_top 的 result_op[1:0]，即 ALU 产生的结果的低两位
state	IN	3	状态机当前的状态，来自 mycpu_top 的 current_state
MemRead	OUT	1	内存读控制信号
RegWrite	OUT	1	写寄存器控制信号
MemWrite	OUT	1	内存写控制信号
RegDst	OUT	2	控制写寄存器的地址
ALUSrc	OUT	2	控制 ALU 的第二个输入值
MemtoReg	OUT	4	控制写寄存器的内容
Branch	OUT	3	标注不同的分支指令
ALUOp	OUT	3	控制 ALU 的操作类型
Jump	OUT	2	标注不同的跳转指令
Write_strb	OUT	4	内存字节写使能信号
PCWrite	OUT	1	控制 PC 的更新

3、结构设计图及功能描述

该部分主题部分为一系列多路选择器，根据输入信号对每个输出信号赋值。因此只 MemWrite 信号为例画出结构设计图。

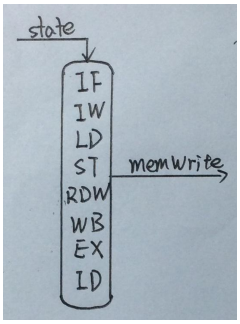


图 2：判断 MemWrite 结构图

该模块根据输入的各种信号，如当前状态、取到的指令等，产生各种的控制信号。根据这些控制信号，CPU 可以决定接下来的操作，例如读写内存，读写寄存器，ALU 应该进行什么运算，是否有跳转等等。

（四）reg_file 模块设计

1、工作原理

该模块实现了包含 32 个 32-bit 通用寄存器的寄存器堆，支持 2 读 1 写端口，0 号寄存器内的值恒为 0，并且同步写、异步读。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号，来自 mycpu_top 的输出时钟
resetn	IN	1	复位信号，低电平同步
waddr	IN	5	写口地址，来自 mycpu_top 的 WriteRegAdd
raddr1	IN	5	读口地址 1，来自 mycpu_top 的 Instruction[25:21]
raddr2	IN	5	读口地址 2，来自 mycpu_top 的 Instruction[20:16]
wen	IN	1	写使能，来自 mycpu_top 的 RegWrite

名称	方向	位宽	功能描述
wdata	IN	32	写数据，来自 mycpu_top 的 WriteData_Reg
rdata1	OUT	32	读口 1 数据，连接到 mycpu_top 的 ReadData1
rdata2	OUT	32	读口 2 数据，连接到 mycpu_top 的 ReadData2

3、功能描述

首先定义了 32 个 32-bit 寄存器 r [31:0]，为实现“同步写、异步读”的功能，读数据部分采用组合逻辑，写数据部分采用时序逻辑。

（五）ALU 模块设计

1、工作原理

该模块用于实现 CPU 需要进行的各种运算，mycpu_top 中含有两个该模块，一个用于实现数值的计算，另一个用于计算跳转指令需要跳转到的 PC 的值

2、接口定义

名称	方向	位宽	功能描述
A	IN	32	操作数 1
B	IN	32	操作数 2
ALUop	IN	4	运算操作
Overflow	OUT	1	指示有符号数的加减法运算结果是否溢出
CarryOut	OUT	1	指示无符号数的加减法运算结果是否产生进位/借位
Zero	OUT	1	指示运算结果是否为 0
Result	OUT	32	ALU 的输出结果

3、功能描述

首先该模块根据 mycpu 传来的信号 ALUop 来判断进行什么操作。因为减法操作可以通过补码来实现，因此 A-B 可以用 A+~B+1 来实现。对于 Overflow，CarryOut 和 Zero 的判断可以通过对结果进行分析得到。

三、实验过程

（一）实验流水账

时间	记录
9 月 8 日 18：00~21:00	一、将之前的多周期 cpu 的接口进行更改，删除之前的握手信号，对新的信号进行赋值； 二、更改 verilog 代码风格，例如将大量的组合逻辑的 always 语句改写为 assign 语句，代码整体更加美观。
9 月 14 日 22：30~23:00	设计总体思路，大致画出总体的结构图。修改代码中不合理之处。
9 月 15 日 23：10~次日 2:00	一、进行行为仿真，根据波形，golden_trace 和 test.s 修改代码，最终行为仿真仍未通过；
9 月 16 日 13：50~17:00	一、进行行为仿真，根据波形，golden_trace 和 test.s 修改代码，最终行为仿真通过； 二、上板验证通过。
9 月 17 日	一、将 cpu_control 模块从时序逻辑改为组合逻辑，调试行为仿真通过，并修改代码风格；

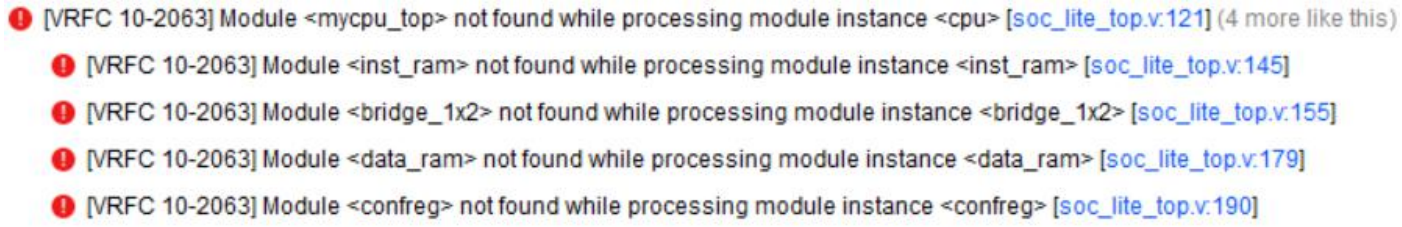
23：00~次日 2:20	二、上板验证通过。

（二）错误记录

1、错误 1

（1）错误现象

第一次进行仿真的过程中报如下错误：



（2）分析定位过程

错误信息显示 mycpu_top 模块没找到，应该是顶层调用的模块的名称和我写的模块的名称不同。于是打开顶层模块寻找其命名。

（3）错误原因

发现顶层中调用的模块命名为 mycpu_top，而我写的名称为 myCPU，因此会报错。

（4）修正效果

将我写的 CPU 模块的名称改为 mycpu_top，之后进行仿真，发现错误信息消失。

（5）归纳总结

在写上层调用的子模块时，要注意其命名，要与上层中的调用一致。

2、错误 2

（1）错误现象

刚开始进行进行仿真，控制台就打印出的 Error。

（2）分析定位过程

通过与 test.s 对比发现，状态机的状态跳转很混乱，于是我认为应该是状态机那里写错了。但是仔细检查代码并没有发现错误！于是认真观察后发现是状态机几个状态名称的参数错误！

（3）错误原因

mycpu_top 模块和 cpu_control 模块的状态机状态名称对应的参数不相同。原因是之前我修改过 mycpu_top 中的参数而忘记修改 cpu_control 中的参数。

（4）修正效果

将参数修改一致后状态机正常跳转。

（5）归纳总结

修改过一个模块中的参数过后，如果其他模块中也有这个参数一定要将其他模块中的参数也修改。

3、错误 3

（1）错误现象

```
[  2277 ns] Error!!!
reference: PC = 0xbfc00390, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfaff008
mycpu    : PC = 0xbfc00390, wb_rf_wnum = 0x04, wb_rf_wdata = 0x00000008
```

进行仿真过程中控制台打印出上图的 Error，然而打开波形发现 wb_rf_wdata 并没有出错。

（4）分析定位过程

继续向后仿真发现所有的错误都是这样:wb_rf_wdata 低字节正确，高三位的字节为 0。打开 mycpu_tb.v 找 debug_wb_rf_wdata_v 找该信号的赋值部分，发现代码如下：

```
wire [31:0] debug_wb_rf_wdata_v;
assign debug_wb_rf_wdata_v[31:24] = debug_wb_rf_wdata[31:24] & {8{debug_wb_rf_wen[3]}};
assign debug_wb_rf_wdata_v[23:16] = debug_wb_rf_wdata[23:16] & {8{debug_wb_rf_wen[2]}};
assign debug_wb_rf_wdata_v[15: 8] = debug_wb_rf_wdata[15: 8] & {8{debug_wb_rf_wen[1]}};
assign debug_wb_rf_wdata_v[7 : 0] = debug_wb_rf_wdata[7 : 0] & {8{debug_wb_rf_wen[0]}};
```

debug_wb_rf_wdata_v 该信号与 debug_wb_rf_wen 也有关，于是错误原因可能在 debug_wb_rf_wdata。

（5）错误原因

最后发现 debug_wb_rf_wen 信号位宽为 4，然而只给其赋值为 1’b1，这也就解释了为什么只有低字节正确。

（4）修正效果

将 wb_rf_wdata 从 1’b1 改为 4’b1111 之后，发现 debug_wb_rf_wdata 的值与 wb_rf_wdata 相同。

（5）归纳总结

在对信号进行赋值时，要主要信号的位宽。

4、错误 4

（1）错误现象

控制栏报出 PC 错误

（2）分析定位过程

向前推几个指令发现是一个 jr 指令跳转错误，PC 并没有跳到正确的地方。PC 本应该要变为到 31 号寄存器内的值，我发现 PC 确实取到了 31 号寄存器内的值，但是这个值与正确的值并不一样。发现 31 号寄存器从开始存入的值就是错的。

（3）错误原因

jal 指令执行错误，存入了 PC 跳转之后的地址，原因是其在存入时，PC 已经跳转。于是加入一个寄存器，保存 PC 跳转之前的值，将这个值存入到 31 号寄存器中。

（4）修正效果

修改过之后错误顺利解决。

（5）归纳总结

该类问题属于代码逻辑错误，需完善代码逻辑。

5、错误 5

（1）错误现象

PC 的值一直为 0xbfc00000，不发生变化。

（2）分析定位过程

发现 PCWright 一直为 0，所以 PC 不能更新。然而，PCWright 是根据状态赋值的，当状态为 WB 时，PCWright 为 1，但是 mycpu_top 里面的 state 是正确跳转的。于是查看 cpu_control 里面的 state，发现是 Z。肯定是因为接口处有问题。

（3）错误原因

mycpu_top 里 state 位宽为 3，而 cpu_control 里 state 里定义为 4 位。这是因为之前删有一个无用状态，在 mycpu_state 里将位宽改为 3，而 cpu_control 里未更改。

（4）修正效果

将 state 位宽修改过之后错误顺利解决。

四、实验总结

到目前为止，我认为最大的收获就是，通过本次实验，我基本掌握了更加规范的 verilog 代码格式，并修改了之前的代码。并且通过这个过程，让我对 Verilog 语言有了更加清晰的认识，我更加清楚地了解到 Verilog 语言和其他语言(如 C 语言)的区别，Verilog 语言是用来描述电路的，描述一个客观存在的实体，而不是描述逻辑，因此 Verilog 语言更像是搭积木。

并且，第一个实验已经让我熬夜调波形，可想而知以后还会有多少个这样的夜晚.....所以第一个实验也算是让我有个心理准备吧。