Lab 3 报告

学号 2020K8009929013
姓名史文轩
箱子号 69

一、实验任务（10%）

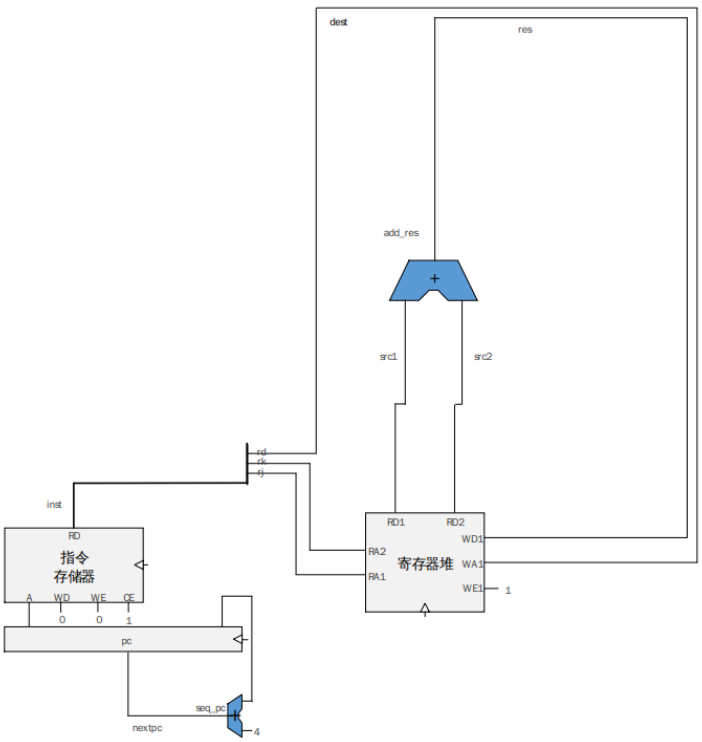
本次实验的要求是实现一个支持 20 条指定 LoongArch32 精简版指令的单周期 CPU（主体代码已经给出），通过仿真波形调试修正代码中的错误，与实验中提供好的 golden cpu 对比并进行上板验证，以通过所有仿真任务。

二、实验设计（40%）

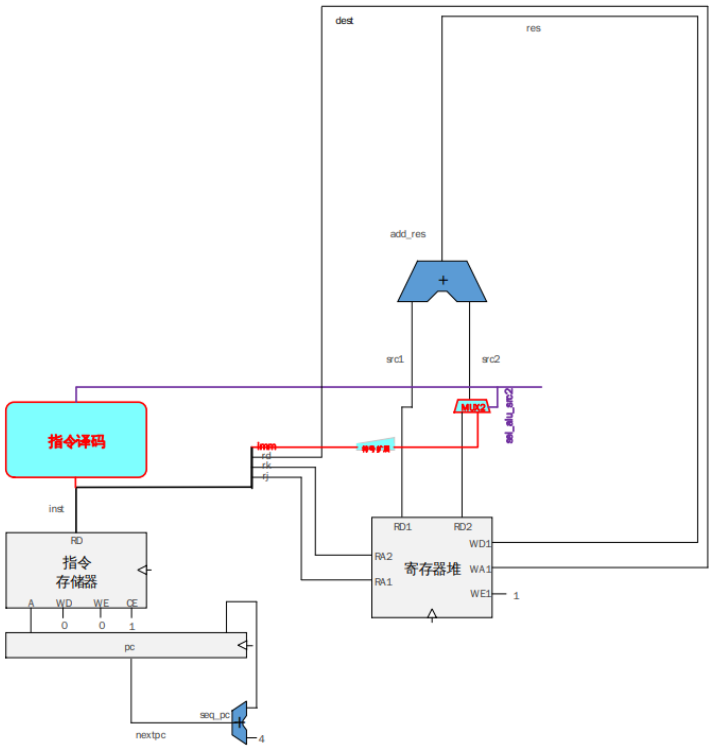
（一）总体设计思路

单周期 CPU 本质上是一个数字逻辑电路，因而我们在设计单周期 CPU 时采用“先设计数据通路，再确定控制逻辑”的思路。具体实现时就是根据每一条指令要实现的功能画出需要的数据通路，每新增一条指令就把新指令的数据通路加到原来的数据通路上。能复用的尽量复用，不能复用的新进入进去，同一接口有冲突的用多路选择器选择并新增选择信号。这样一条一条指令加入，最终就能实现所有要求的指令。

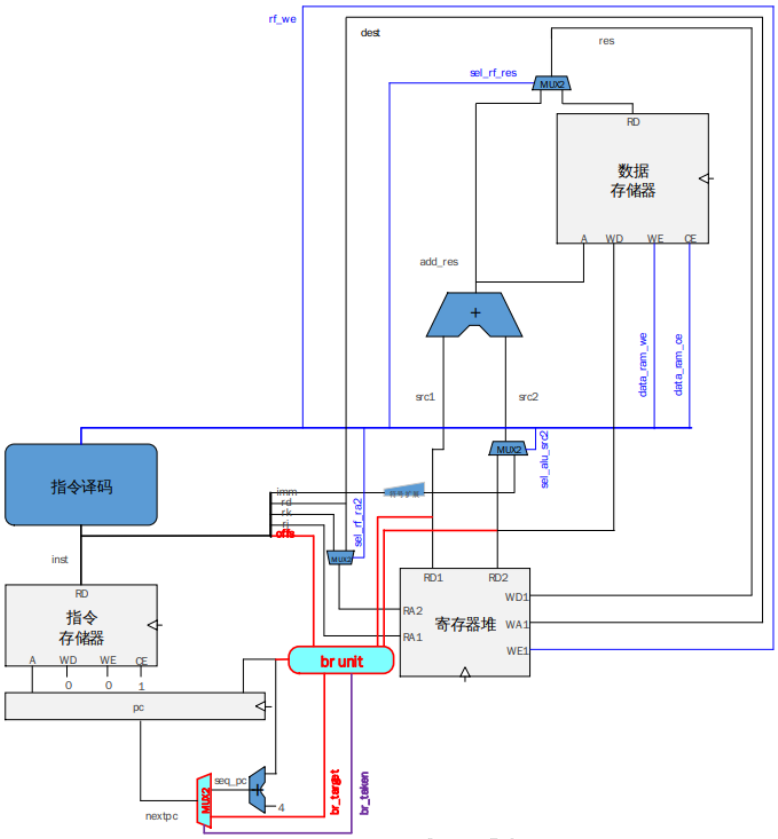
比如要实现 add.w 指令，除了有关 PC 和指令更新的部分，我们还要考虑到 add.w 要从寄存器堆里读出两个操作数，也就是在译码后需要接入寄存器堆的两个读地址，从读出数据接口将两个操作数接入加法器，输出结果再配合译码得到的寄存器地址写入指定寄存器中。仅为实现这一条指令我们就可以画出如下的数据通路：



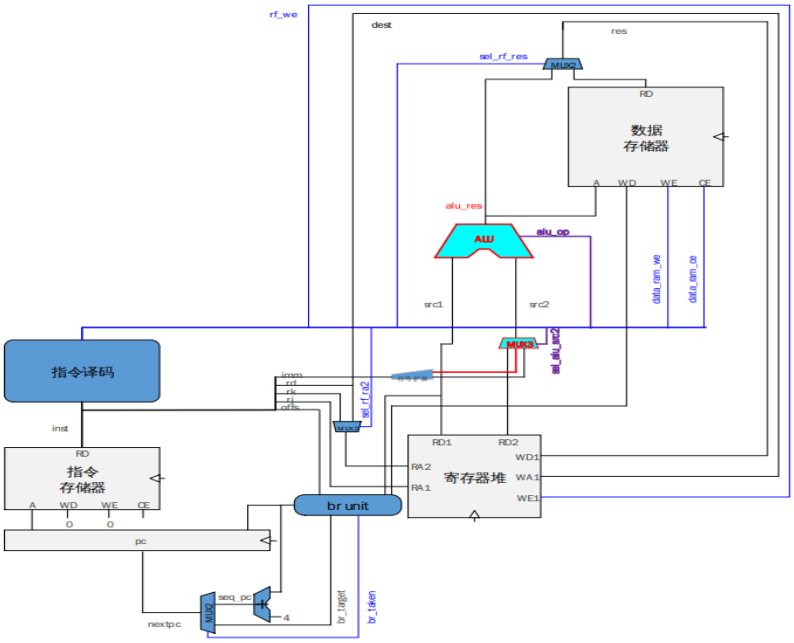
然后我们考虑 addi.w 指令，为实现该指令的操作，上图的大部分部件都可以复用，只需要再加一个对立即数的符号扩展部件。同时指令变多后还要新加入一个指令译码器来区分每条指令完成的操作。因而可作下图：



同理，我们要新实现 ld.w 指令就要在数据通路中加入数据存储器以读取数据，同时取回的数据与加法器的结果都是要写进寄存器堆的，这样的冲突可以用加一个二选一的选择器解决，选择信号可以直接由译码得来。与之类似的，st.w 指令不需要新的部件，但是为了区分其与 ld.w 指令的功能，需要用不同的控制信号来决定数据通路的操作。还有如 bne 这类跳转指令，需要对 PC 的跳转方式做判断，因而也可以加一个多路选择器，指令译码时可以译码出偏移量这种信号。实现这五条基础指令的数据通路已经距离我们的二十条指令非常近了。

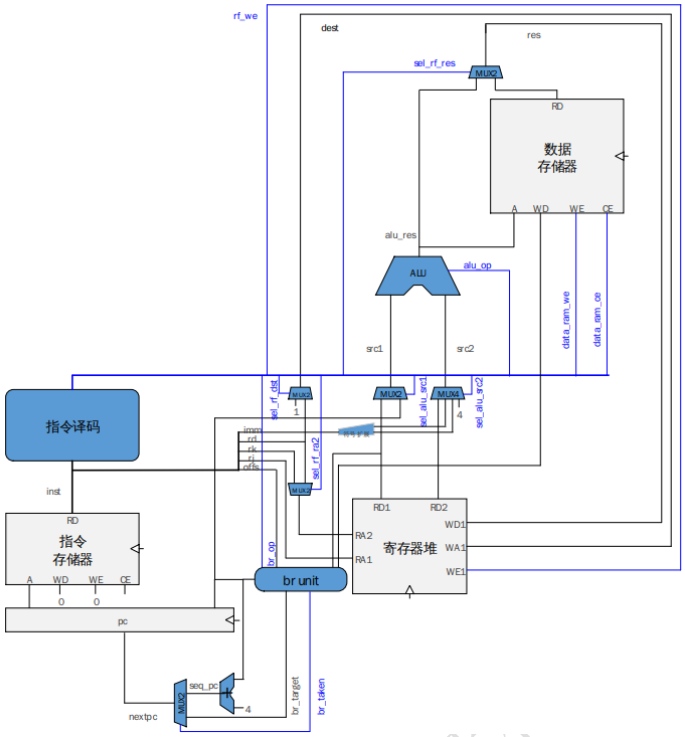


通过对比要实现的二十条指令与上述五条指令，我们可以发现大部分多出的指令都是计算指令。要实现这些计算指令，需作出的改变是将加法器升级为ALU部件，以支持新增ALU类指令的算术逻辑功能，同时需新增ALU部件的控制信号 alu_op。同时由于移位指令和 lu12i.w 指令的立即数位宽改变，可以通过符号扩展或者增加新的数据通路解决。数据通路更改如下：



除了计算类指令增加，新增的指令中还包含四条转移类指令：beq、b、bl、jirl.这几条跳转指令的功能大体一致，数据通路大都可以复用，针对不一样的操作可以用控制信号将其区分开来。讲这些指令的数据通路都加入进

去，并对控制信号进行了一定的调整后，得到了最终的数据通路和控制信号图。



（二）重要模块 1 设计：寄存器堆

1、工作原理

输入读地址，可将对应地址的寄存器内值从读出口读出；输入写地址和写数据，在读使能为 1 时可以将数据写入对应地址的寄存器。

2、接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
raddr1	IN	5	读地址 1
rdata1	OUT	32	读出数据 1
raddr2	IN	5	读地址 2
rdata2	OUT	32	读出数据 2
we	IN	1	写使能
waddr	IN	5	写地址
wdata	IN	32	写数据

3、功能描述

写功能是同步写，需用到时序逻辑，时钟上升沿且写使能为 1 时写入；两个读完全等价，都是异步写，用二路选择器保证零号寄存器读出的值总为 0，其他的正常读出。

（三）重要模块 2 设计：ALU

1、工作原理

输入两个操作数和操作码（用于选择计算的方式），可通过计算直接输出指令类型的计算结果。

2、接口定义

名称	方向	位宽	功能描述
alu_op	IN	12	计算操作码，用于选择计算类型
alu_src1	IN	32	操作数 1
alu_src2	IN	32	操作数 2
alu_result	OUT	32	计算结果

3、功能描述

集成了包括加、减、与、或、逻辑左移等十二种计算方式，用组合逻辑的方式把十二种结果都计算出来，然后以计算操作码为选择信号建立多路选择器，选择出对应的计算结果。

三、实验过程（50%）

（一）实验流水账

周三仔细阅读了讲义的相关部分，配置好了本次实验的实验环境。

周四通过仿真不断调试，被一个时序上的 bug 卡壳难以往下进行。

周五发现问题所在，并找出了剩下的错误，通过了仿真和上板检验。

周六书写实验报告。

（二）错误记录

1、错误 1：变量未声明

（1）错误现象

无法正常调出波形并在 tcl console 中报错如下：

```
ERROR: [VRFC 10-2989] 'fs_pc' is not declared [D:/calab_project/cdp_ede_local-master/cdp_ede_local-master/dc_
ERROR: [VRFC 10-2989] 'ds_valid' is not declared [D:/calab_project/cdp_ede_local-master/cdp_ede_local-master/
ERROR: [VRFC 10-2989] 'ds_pc' is not declared [D:/calab_project/cdp_ede_local-master/cdp_ede_local-master/dc_
ERROR: [VRFC 10-2989] 'rfrom_mem' is not declared [D:/calab_project/cdp_ede_local-master/cdp_ede_local-maste
ERROR: [VRFC 10-2865] module 'mycpu_top' ignored due to previous errors [D:/calab_project/cdp_ede_local-maste
```

（2）分析定位过程

Tcl console 给出的报错信息很明显，直接在代码中定位并改为声明过的代码即可。

（3）错误原因

变量未声明

（4）修正效果

将未声明的变量根据逻辑选择直接删去或者改为生命过的变量，发现该报错消失，可以观察波形。

2、错误 2：第一个时钟周期内寄存器堆的写数据错误

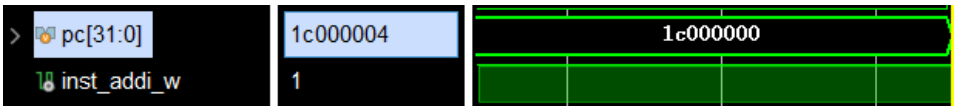
（1）错误现象

Tcl console 中报错如下：

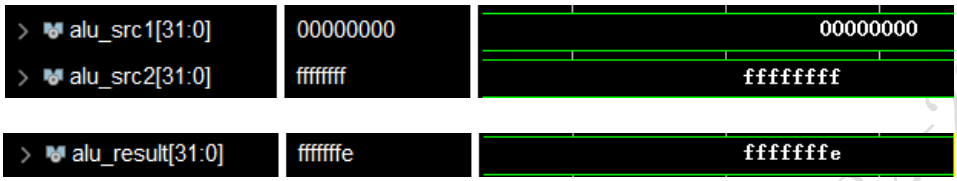
```
reference: PC = 0x1c000000, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xffffffff
mycpu    : PC = 0x1c000000, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000
```

（2）分析定位过程

先确定该条指令的指令类型为 addi.w 指令：



查看 ALU 的输入发现计算操作码和操作数都正确但结果错误：



因而想到可能是调用 ALU 时出错，定位到此：

```
alu u_alu(
    .alu_op     (alu_op    ),
    .alu_src1   (alu_src2  ),
    .alu_src2   (alu_src2  ),
    .alu_result (alu_result)
    );
```

（3）错误原因

调用 alu 模块时变量输入错误。

（4）修正效果

该错误消失，此 PC 下寄存器的写数据正确。

3、错误 3：PC 值错误

（1）错误现象

Tcl console 中报错如下：

```
[   2017 ns] Error!!!
    reference: PC = 0x1c000004, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xffffffff
    mycpu    : PC = 0x1c000000, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xffffffff
```

（2）分析定位过程

这个错误困扰了我很长时间，由于这是初态 PC 值顺序跳转到下一个 PC 值的错误，因而只可能是给出的顺序跳转时刻比用于比对的跳转时刻慢一拍，但由于这个错误并不常见，因而一开始不是很敢下手。最终定位在跳转用的的 reset 信号上。

```
always @(posedge clk) reset <= ~resetn;
```

（3）错误原因

跳转时序慢一拍。

（4）修正效果

将上述给 reset 信号赋值的时序逻辑改为组合逻辑（并改变了 reset 变量的变量类型）。发现上述时序问题消失，进入下一个错误。

```
wire         reset;
//always @(posedge clk) r
assign reset = ~resetn;
```

4、错误 4：bl 指令没有进行寄存器的写操作

（1）错误现象

在 tcl console 中报错如下：

```
[   2247 ns] Error!!!
    reference: PC = 0x1c000198, wb_rf_wnum = 0x01, wb_rf_wdata = 0x1c00019c
    mycpu    : PC = 0x1c007618, wb_rf_wnum = 0x17, wb_rf_wdata = 0x00000001
```

（2）分析定位过程

这个错误的报错容易给人误导，由于 reference 的显示的状态与 mycpu 的前一个状态完全符合，导致我认为是错误三更正引发的时序往前了一拍，因此苦恼许久。后经过同学提醒，reference 的底层逻辑是只有存在寄存器写的操作时才会更新，因而我很自然想到会不会是这条 bl 指令没有进行寄存器写操作呢？

在查阅代码后，发现 rf_we 信号是由 gr_we 信号赋值的，又有下面的逻辑：

```
assign gr_we         = ~inst_st_w & ~inst_beq & ~inst_bne & ~inst_b & ~inst_bl;
```

~inst_bl 就是错误的根源。

（3）错误原因

bl 指令没有进行写操作。

（4）修正效果

去掉&~inst_bl 之后发现该错误消失，进入下一个错误。

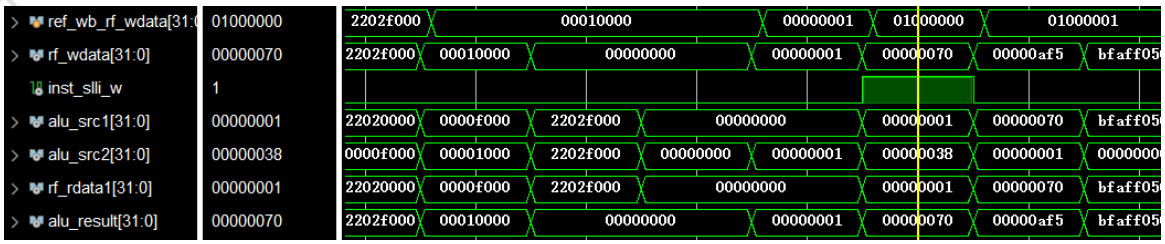
5、错误 5：移位操作出错

（1）错误现象

在进行一条移位操作时报错。

（2）分析定位过程

经过查看波形，发现是移位操作的两个操作数都是正确的，但 ALU 计算出的移位操作结果不正确：



可以很明显地看出，如果是将一号操作数（1）左移无论如何不会出现（0070）这种有多个一的情况出现。因此，我去查找了 ALU 关于移位操作的底层实现，发现代码如下：

```
// SLL result
assign sll_result = alu_src2 << alu_src1[4:0];
```

显然，两个操作数的位置颠倒了。

（3）错误原因

两个操作数的位置颠倒导致的 ALU 中逻辑左移操作结果出错。

（4）修正效果

将上述错误改正过来之后，发现该报错消失，开始报下一个错误。

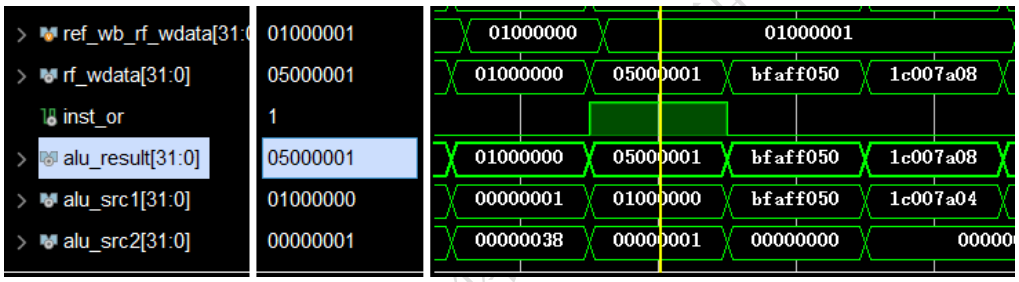
6、错误 6：或操作出错

（1）错误现象

在进行一条或操作指令时报错。

（2）分析定位过程

经过查看波形，发现是移位操作的两个操作数都是正确的，但 ALU 计算出的或操作结果不正确：



可以很明显地看出，如果是将一号操作数（01000000）与二号操作数（00000001）逻辑或之后不会出现（05000001）这种结果。因此，我去查找了 ALU 关于或操作的底层实现，发现代码如下：

```
assign or_result  = alu_src1 | alu_src2 | alu_result;
```

正常的或操作显然不该或上 alu_result 这个信号。

（3）错误原因

ALU 中或操作实现结果出错。

（4）修正效果

将上述错误改正过来之后，发现该报错消失，开始报下一个错误。

7、错误 7：逻辑右移操作出错

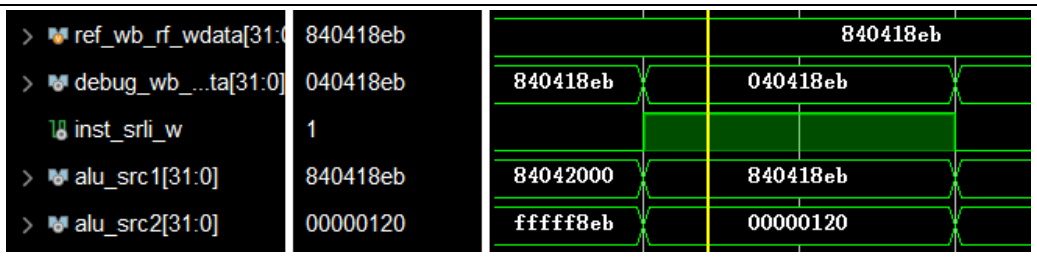
（1）错误现象

在进行一条或逻辑右移指令时报错。

```
[  52897 ns] Error!!!
    reference: PC = 0x1c0011f0, wb_rf_wnum = 0x0a, wb_rf_wdata = 0x840418eb
    mycpu    : PC = 0x1c0011f0, wb_rf_wnum = 0x0a, wb_rf_wdata = 0x040418eb
```

（2）分析定位过程

经过查看波形，发现是移位操作的两个操作数都是正确的，但 ALU 计算出的移位操作结果不正确：



将 debug 信号与 ref 信号进行对比就可以看出二者相差不大，只是正确的结果应该在第 32 位多一个 1。因此，我怀疑可能是 ALU 中有关的操作出了差错，在查找了 ALU 关于逻辑右移操作的底层实现，发现代码如下：

```
assign sr_result   = sr64_result[30:0];
```

只取结果的 31 位显然是不合理的。

（3）错误原因

ALU 中逻辑右移操作的底层实现出错（少赋值一位）导致波形出错。

（4）修正效果

将上述错误改正过来之后，发现该报错消失，测试通过。

四、实验总结（可选）

本次实验总的来说并不算难，除了个别错误有一定误导性之外都是比较明显的错误。但是本次实验提供的实验框架是不简单的，我花费了不少时间也没完全看懂整个系统的实现流程，希望老师能在研讨课上为我们稍微讲解一下。