



单周期CPU设计实验

阶段一

目录

- 01 实验内容
- 02 CPU结构设计方案
- 03 参考设计代码解读
- 04 设计开发环境介绍

目录

- 01 实验内容
- 02 CPU结构设计方案
- 03 参考设计代码解读
- 04 设计开发环境介绍



实验内容

- 实验任务：
 - 完善一个单周期LA32 CPU设计，支持add.w、addi.w、ld.w、st.w和bne指令功能，最终远程上板测试通过。
- 实验步骤：
 - 将 CPU_CDE1.zip 解压到路径上无中文字符的目录里，实验环境为 CPU_CDE1
 - 完善位于CPU_CDE1/mycpu_verify/rtl/mycpu目录下的单周期CPU（共计5处需完善）
 - 打开myCPU工程（CPU_CDE1/mycpu_verify/run_vivado/mycpu_prj1/mycpu.xpr）。
 - 对myCPU工程中的inst_ram重新定制，此时选择对应func的coe文件（CPU_CDE1/func /inst_ram.coe）。
 - 运行myCPU工程的仿真（进入仿真界面后，直接点击 run all），开始调试。打开CPU_CDE1/mycpu_verify/testbench目录下mycpu_tb.sv文件，通过修改dip_sw值观察相应leds的值（每次修改dip_sw值都要重新仿真）。（因为本实验的测试程序为斐波那契数程序，斐波那契数列是：0, 1, 1, 2, 3, 5, ……从第三项开始，每一项都等于前两项之和。规定数列第三项为 $f(1)$ ，即 $f(1)=1$ ， $f(2)=2$ ， $f(3)=3$ ， $f(4)=5$ ，……。修改拨码开关dip_sw值相当于修改 n ，对应的 $f(n)$ 为leds值。
 - myCPU仿真通过后，综合实现后生成bit流文件，进行上板验证

目录

- 01 实验内容
- 02 CPU结构设计方案
- 03 参考设计代码解读
- 04 设计开发环境介绍



务必先明确设计方案，再进行代码实现

- CPU 本质上是一个数字逻辑电路，它的设计遵循数字逻辑电路设计的一般性方法
- 首先设计出数据通路，再确定控制逻辑
 - 数据在电路中流转的路径称为数据通路，如CPU中的运算逻辑、存储单元、输入输出及模块间的连接总线。
 - 数据通路中的多路选择器、时序逻辑器件等包含控制信号，产生这些控制信号的逻辑称为控制逻辑
- CPU设计的特点：根据指令系统规范中的定义设计出 “数据通路 + 控制逻辑”
 - 对指令系统中定义的指令逐条进行功能分解，得到一系列操作和操作的对象，这些操作和操作的对象必然对应其各自的数据通路
 - 因为指令间存在一些相同或相近的操作和操作对象，所以我们可以只设计一套数据通路供多个指令公用
 - 对于确实存在差异无法共享数据通路的情况，只能各自设计一套，再用多路选择器从中选择出所需的结果



待实现指令定义

- **add.w (add word)**
- **addi.w (add immediate word)**
- **ld.w (load word signed)**
- **st.w (store word)**
- **bne (branch on not equal)**

add.w rd, rj, rk $GR[rd] = GR[rj] + GR[rk]$

addi.w rd, rj, si12 $GR[rd] = GR[rj] + sext32(si12)$

ld.w rd, rj, si12 $GR[rd] = MEM[GR[rj] + sext32(si12)][31:0]$

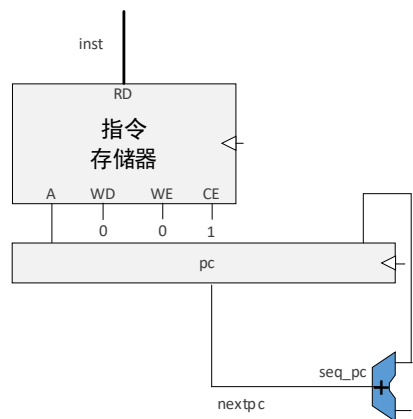
st.w rd, rj, si12 $MEM[GR[rj] + sext32(si12)][31:0] = GR[rd][31:0]$

bne rj, rd, offs16 if $(GR[rj] \neq GR[rd])$ $PC = PC + sext32(\{offs16, 2'b0\})$



支持add.w指令

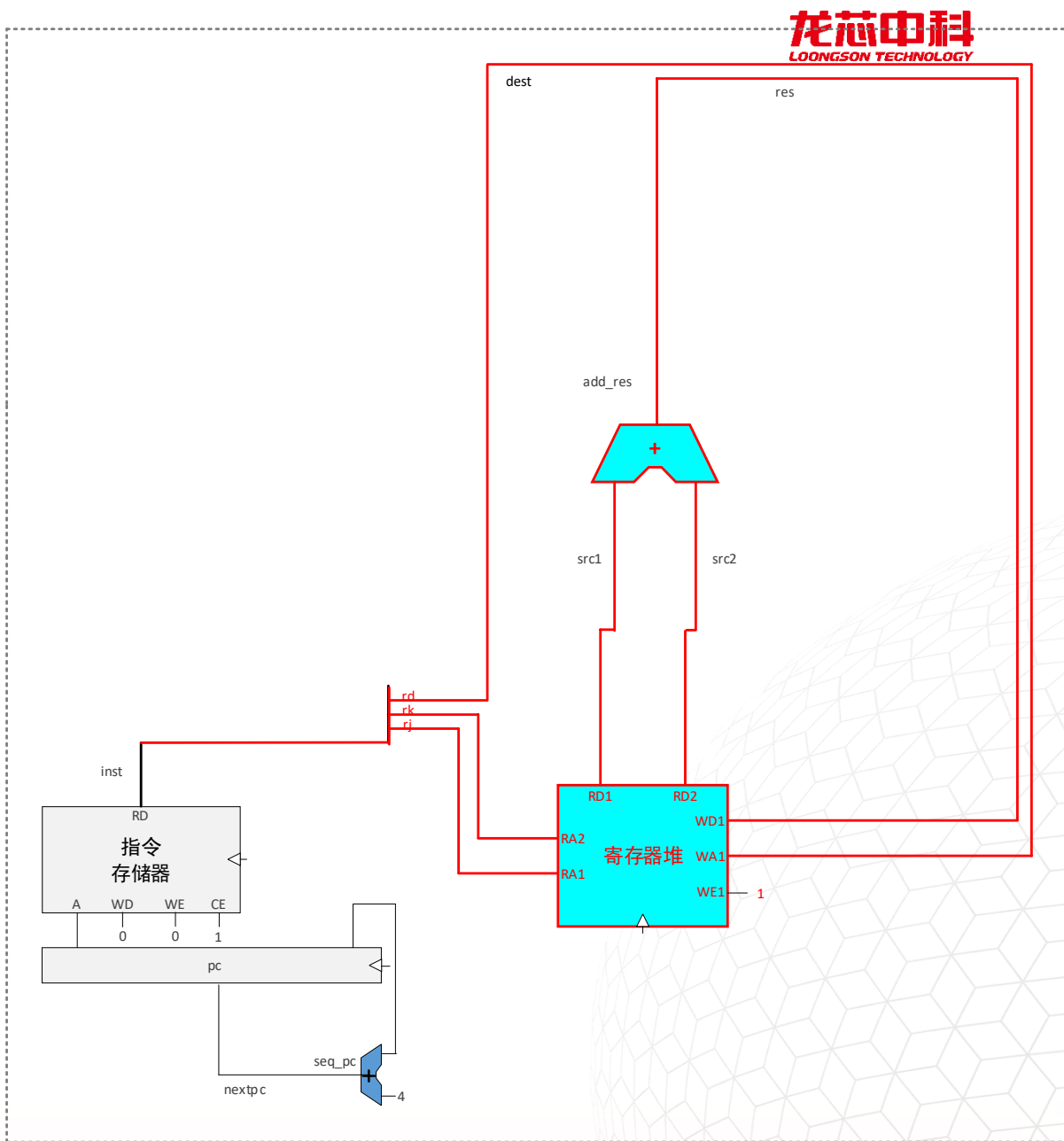
- 所有指令存放在存储器（指令存储器）中
- 指令在指令存储器中的地址是PC
- 每执行完一条指令，继续执行其后面的指令（PC+4）





支持add.w指令

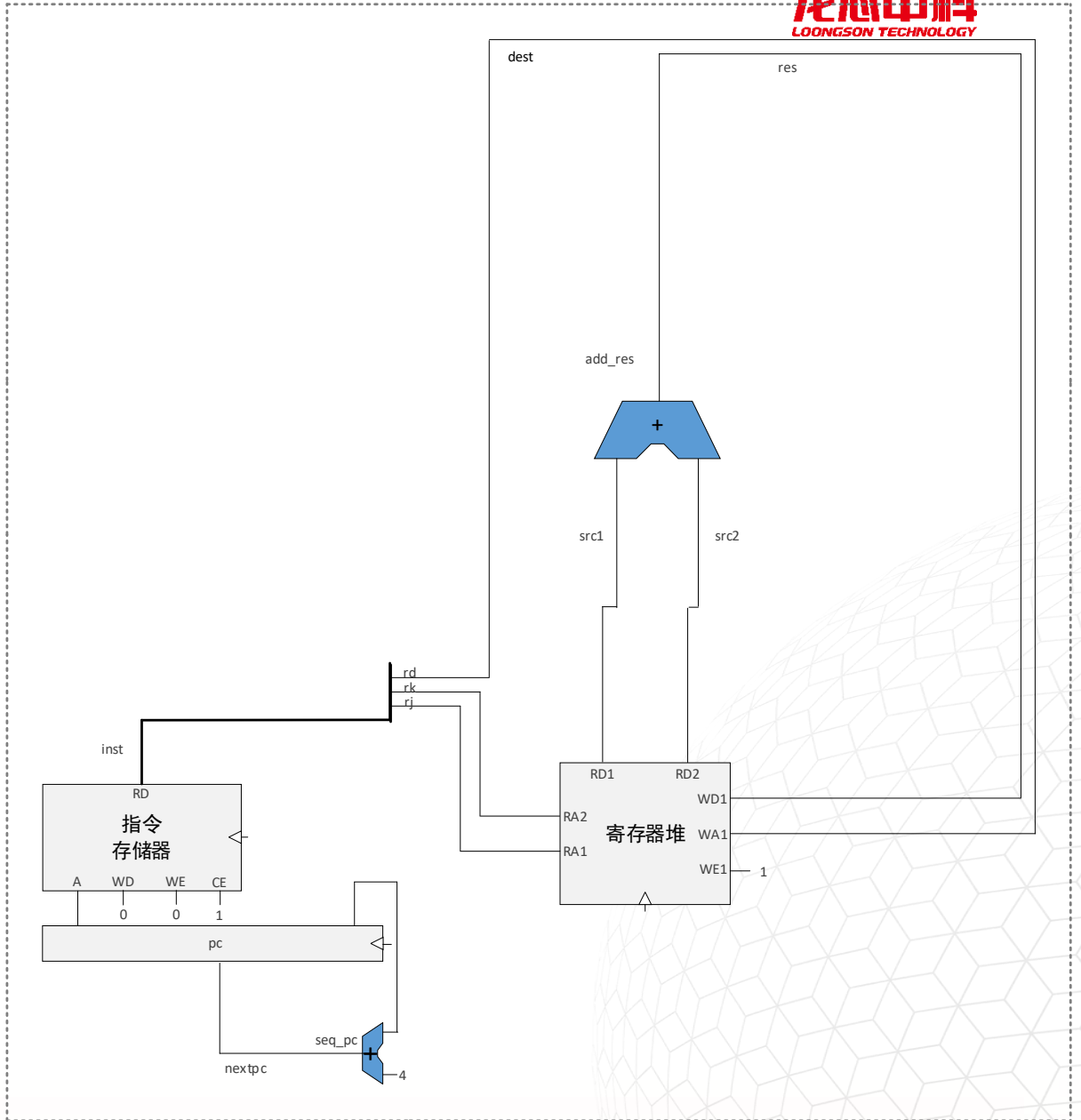
- add.w指令从寄存器堆中读取两个源操作数
- 将两个源操作数相加
- 加法结果写回寄存器堆





支持add.w指令

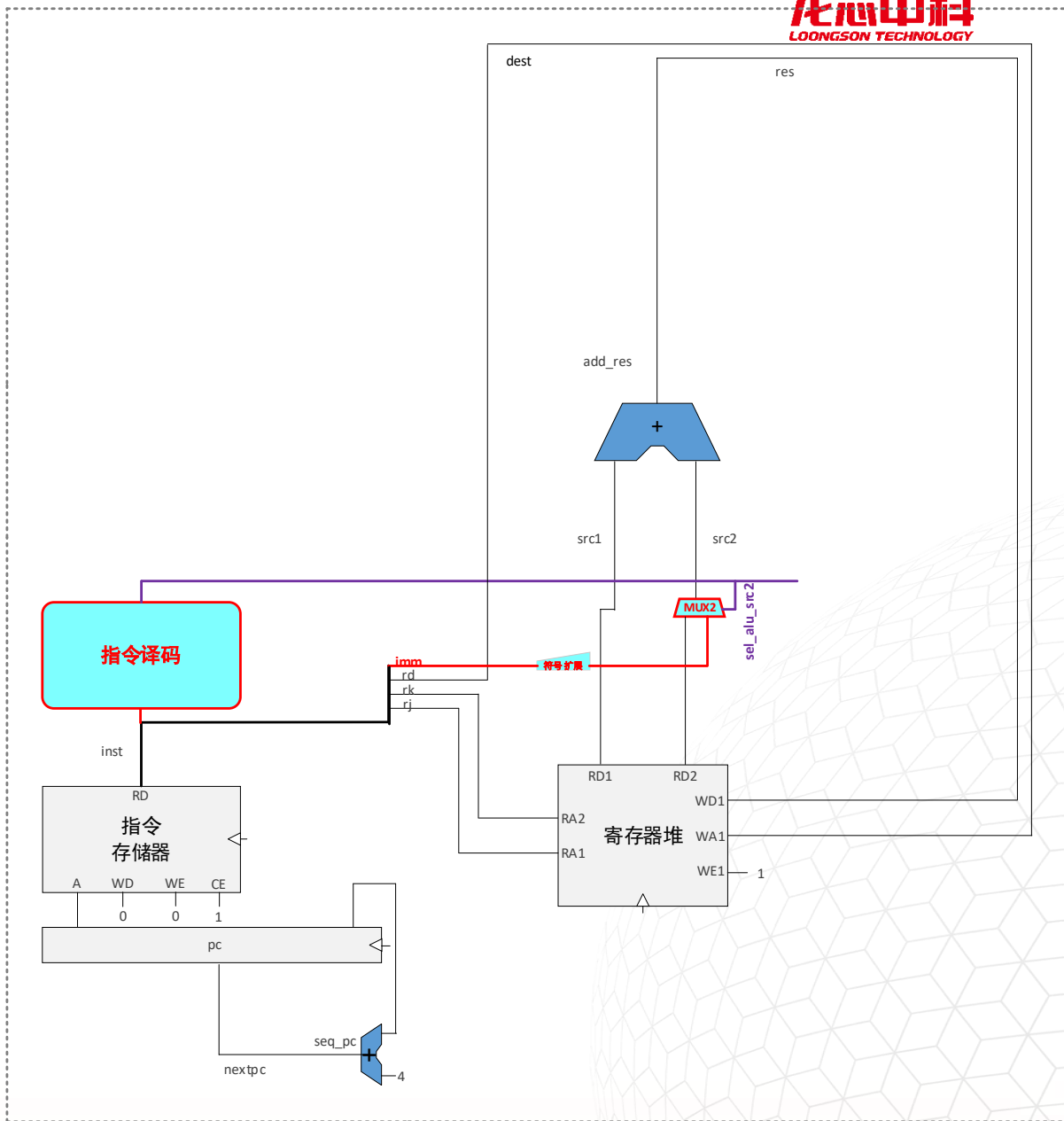
龙芯中科
LOONGSON TECHNOLOGY





支持addi.w指令

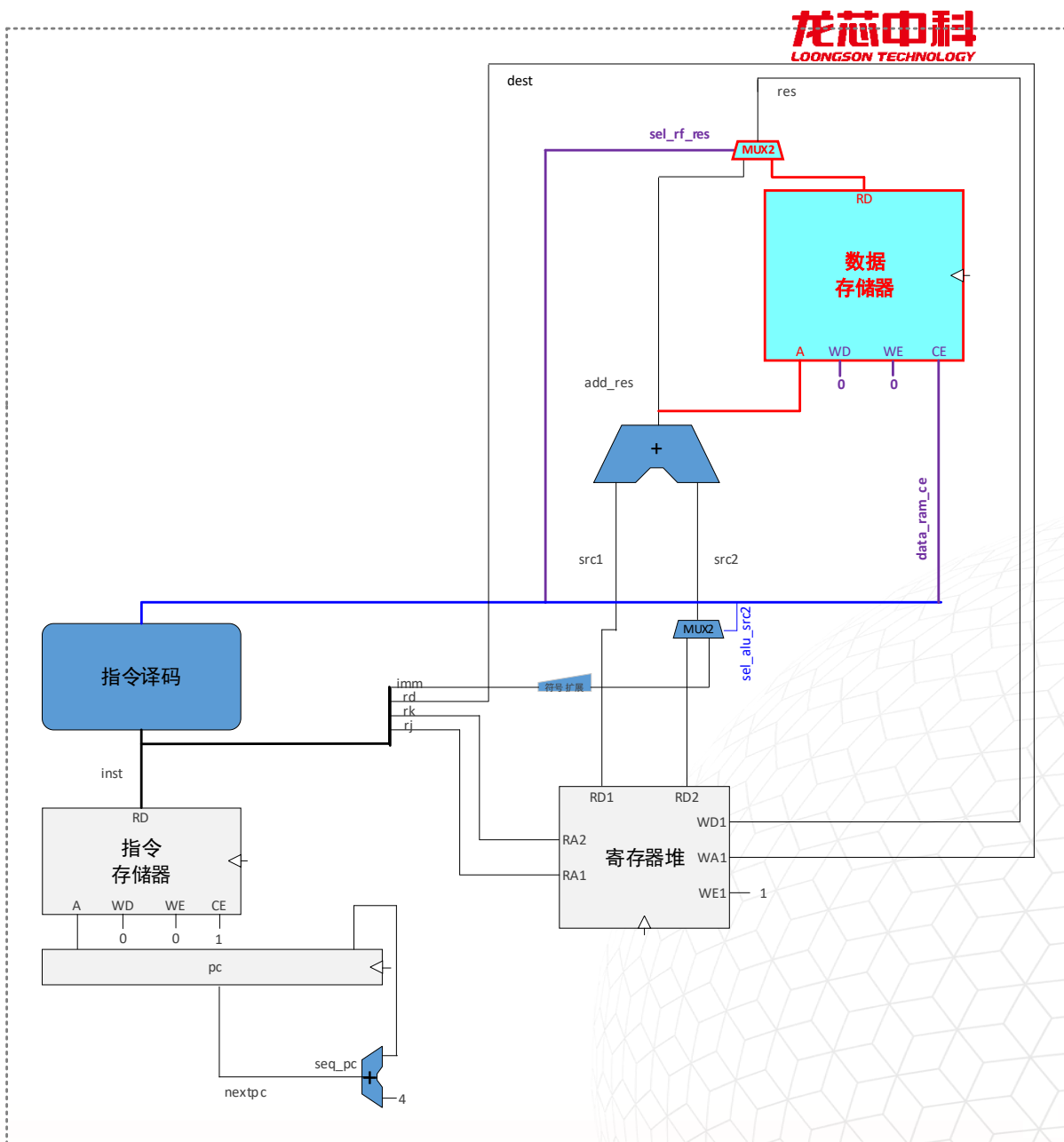
- addi.w指令与add.w指令的差异仅在于相加所用的第2个源操作数不是来自于寄存器堆而是指令中立即数符号扩展至32位





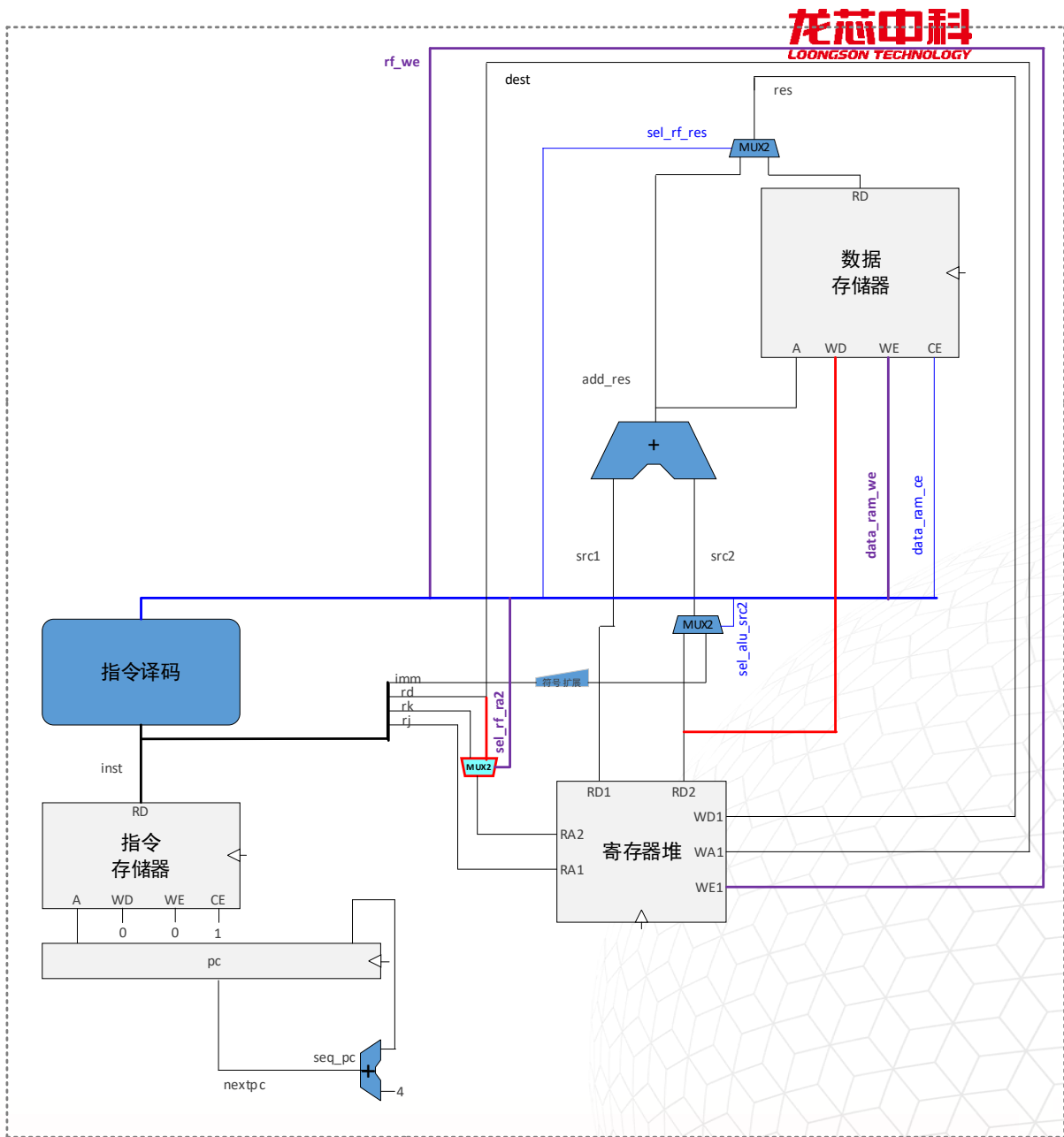
支持ld.w指令

- ld.w指令的地址计算功能可以复用addi.w指令的数据通路
- ld.w指令从数据存储器中读取数据
- 取回的数据与加法器结果二选一，写回寄存器堆





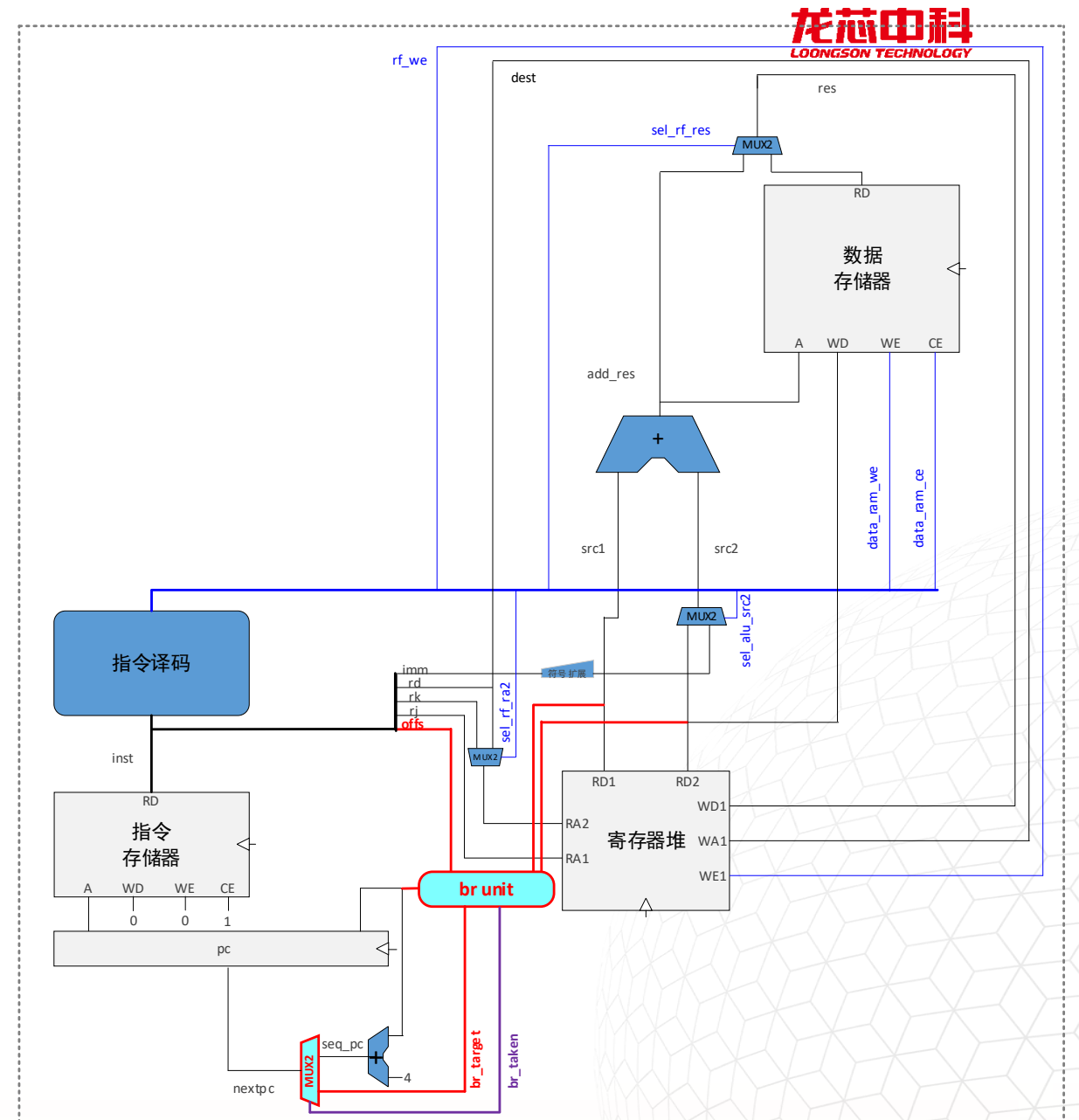
- st. w指令的地址计算可以复用ld. w指令的
- st. w指令写数据存储器
- st. w指令写入数据存储器的值来自于寄存器堆，不过这个源操作数的寄存器号在rd域
- st. w指令不写寄存器堆





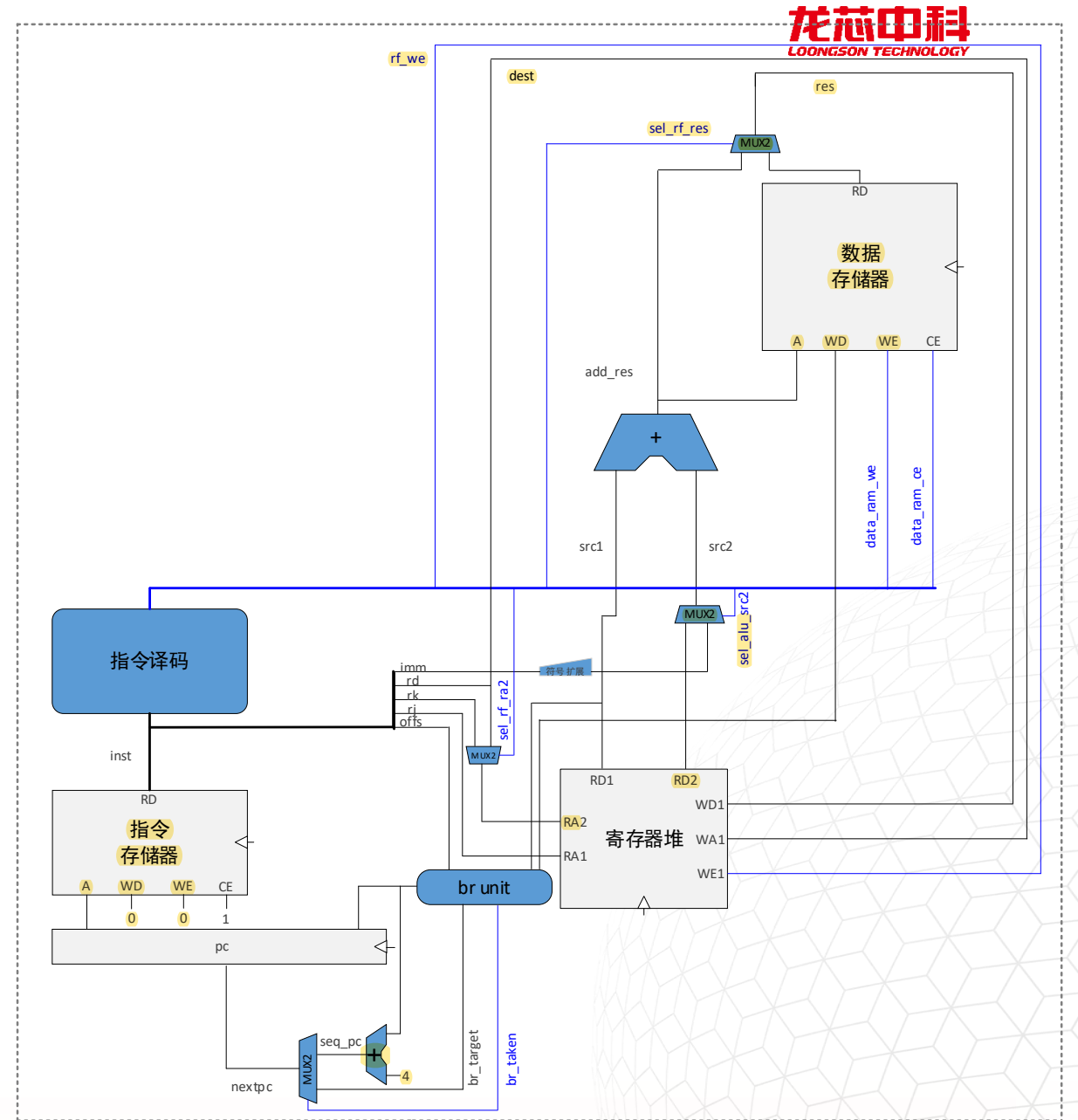
支持bne指令

- bne指令对来自寄存器堆的两个源操作数比较
决定是否跳转 (br_taken)
- bne指令跳转的目标是PC加上指令码中的
offset
- bne指令不跳转时，仍是顺序取指





	add.w	addi.w	ld.w	st.w	bne
sel_rf_ra2	0	0	0	1	0
sel_alu_src2	0	1	1	1	0
data_ram_ce	0	0	1	1	0
data_ram_we	0	0	0	1	0
sel_rf_res	0	0	1	0	0
rf_we	1	1	1	0	0



目录

- 01 实验实验
- 02 CPU结构设计方案
- 03 参考设计代码解读
- 04 设计开发环境介绍



代码与设计

```
module mycpu_top(
```

```
    input          clk,
    input          resetn,
```

```
    output          inst_sram_wen,
    output[31:0]    inst_sram_addr,
    output[31:0]    inst_sram_wdata,
    input [31:0]    inst_sram_rdata,
```

```
    output          data_sram_wen,
    output[31:0]    data_sram_addr,
    output[31:0]    data_sram_wdata,
    input [31:0]    data_sram_rdata
```

```
);
```

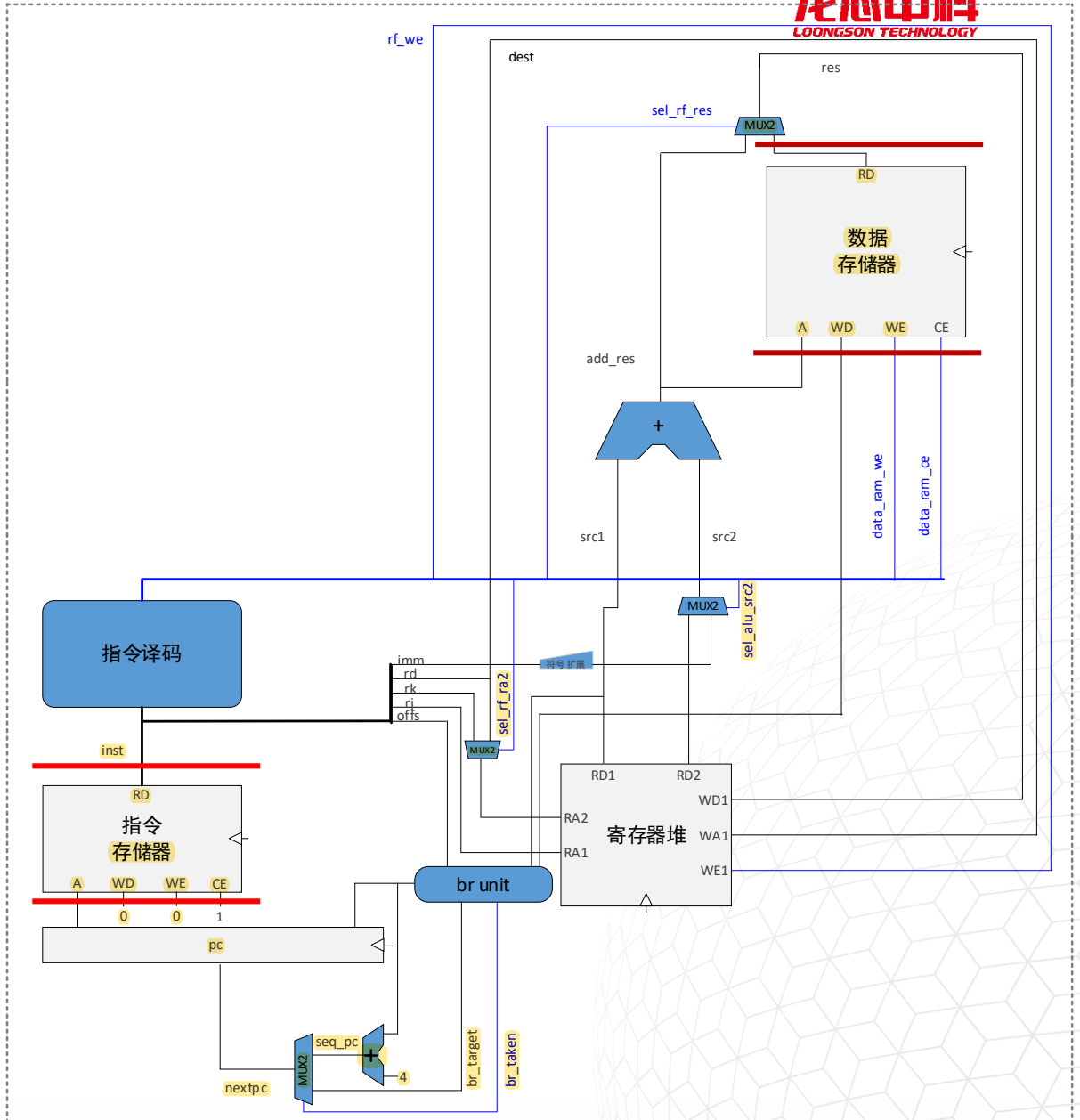
```
.....
```

```
assign inst_sram_wen    = 1'b0;
assign inst_sram_addr   = pc;
assign inst_sram_wdata  = 32'b0;
assign inst              = inst_sram_rdata;
```

```
.....
```

```
assign data_sram_wen=mem_we;
assign data_sram_addr=alu_result;
assign data_sram_wdata=rkd_value;
```

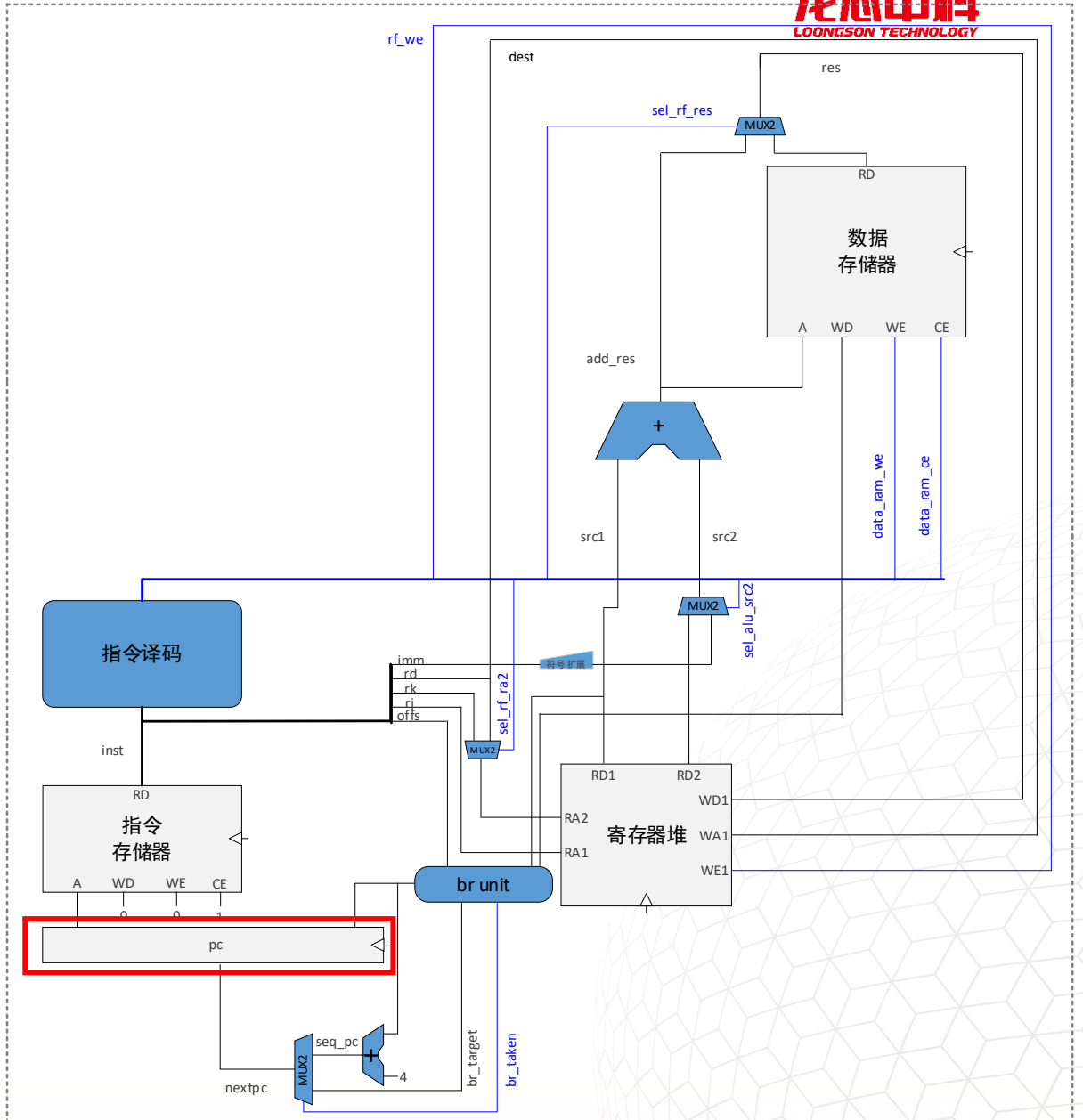
```
.....
```





代码与设计

```
.....  
always@(posedge clk) begin  
    if(!resetsn)begin  
        pc <= 32'h1c000000;  
    end  
    else begin  
        pc <= nextpc;  
    end  
end  
.....
```

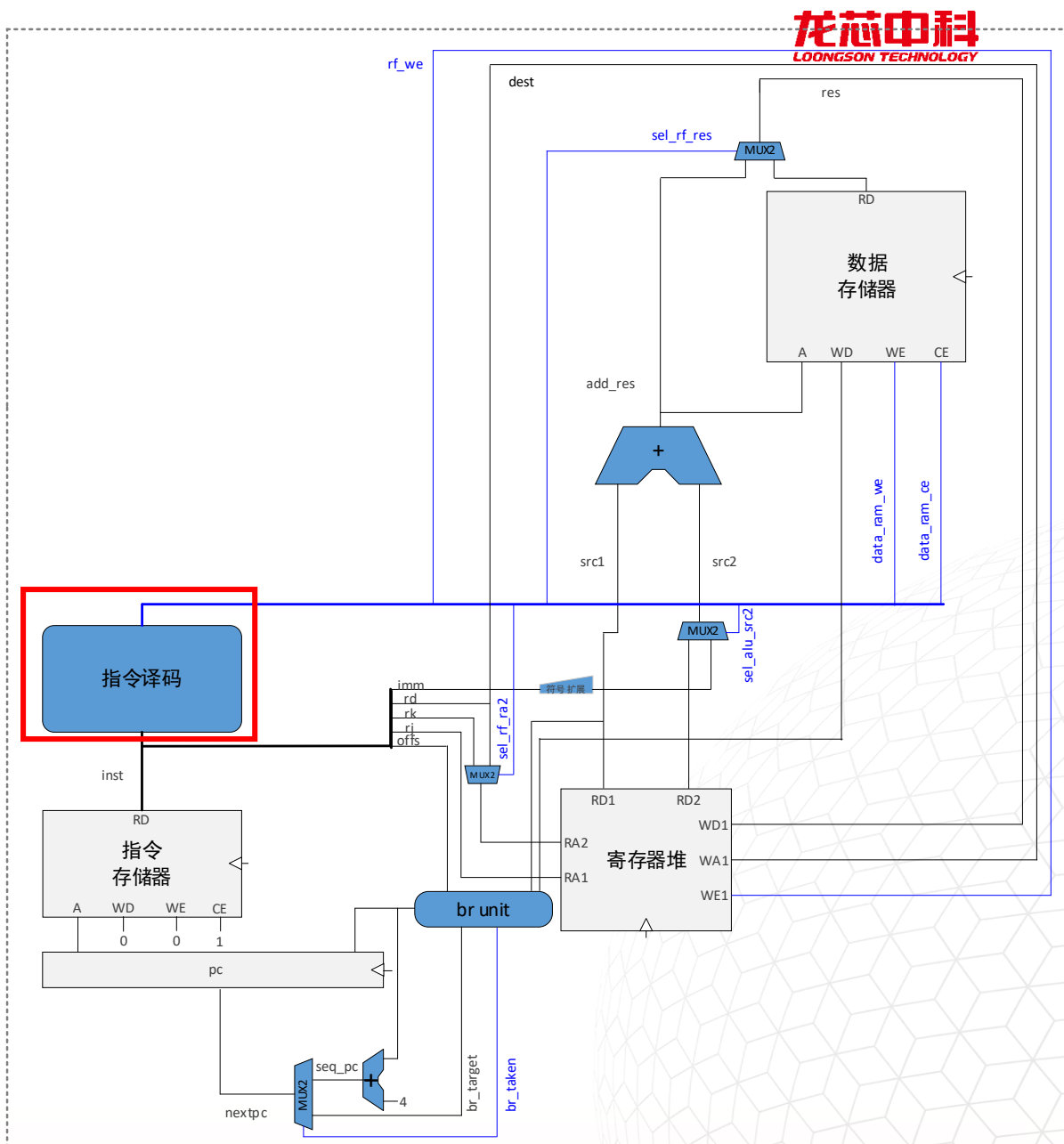




代码与设计

```
assign op_31_26 = inst[31:26];
assign op_25_22 = inst[25:22];
assign op_21_20 = inst[21:20];
assign op_19_15 = inst[19:15];
.....
decoder_6_64 u_dec0(.in(op_31_26 ), .co(op_31_26_d ));
decoder_4_16 u_dec1(.in(op_25_22 ), .co(op_25_22_d ));
decoder_2_4 u_dec2(.in(op_21_20 ), .co(op_21_20_d ));
decoder_5_32 u_dec3(.in(op_19_15 ), .co(op_19_15_d ));
.....
assign inst_add_w = op_31_26_d[6'h00] & op_25_22_d[4'h0]
& op_21_20_d[2'h1] & op_19_15_d[5'h00];
assign inst_addi_w = op_31_26_d[6'h00] & op_25_22_d[4'ha];
assign inst_ld_w = op_31_26_d[6'h0a] & op_25_22_d[4'h2];
assign inst_st_w = //在这里实现inst_st_w指令的译码
assign inst_bne = op_31_26_d[6'h17];

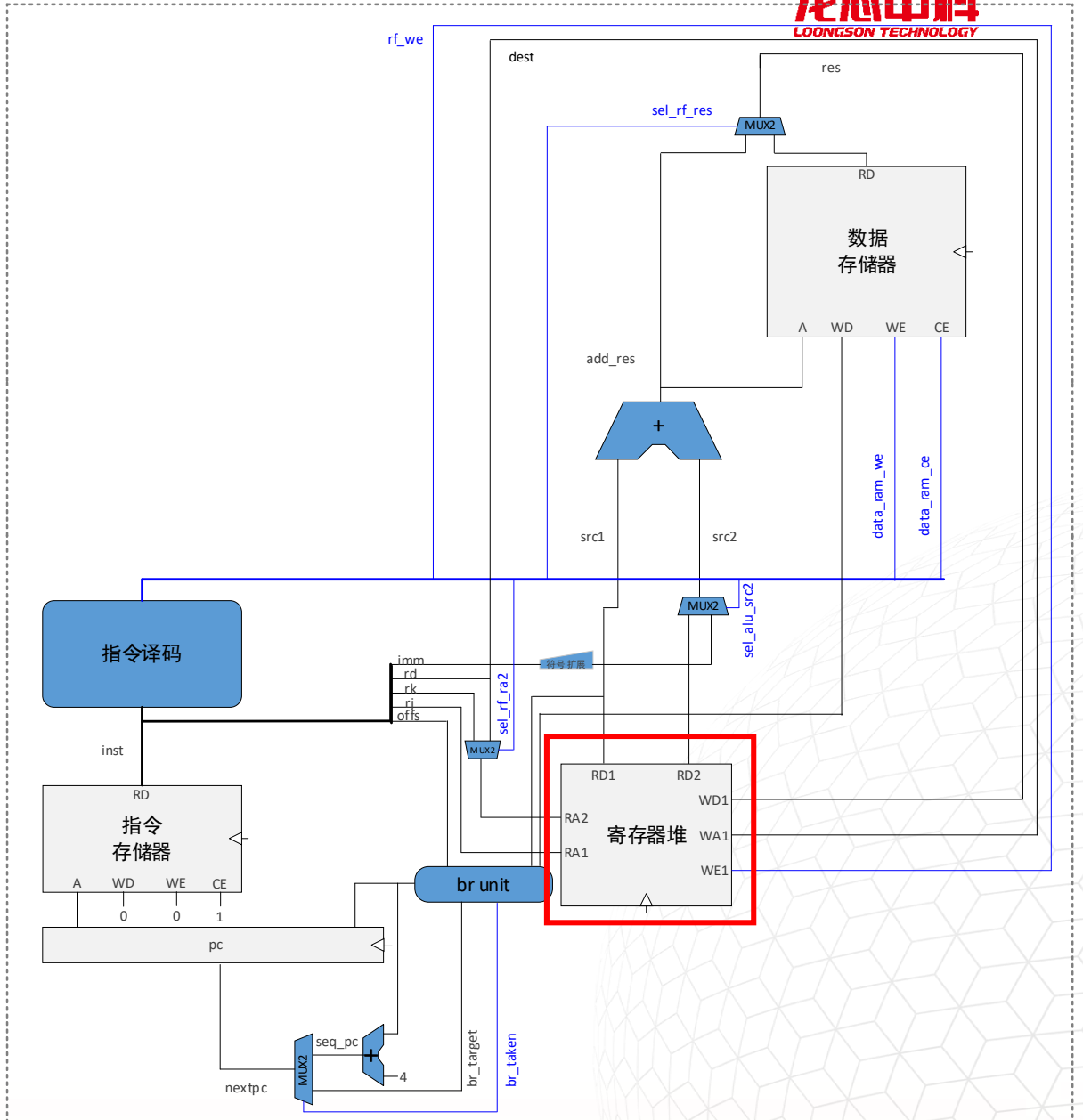
assign src2_is_imm = //在这里实现立即数选择信号
assign res_from_mem = inst_ld_w;
assign gr_we = inst_add_w | inst_ld_w | inst_addi_w;
assign mem_we = inst_st_w;
assign src_reg_is_rd= inst_bne | inst_st_w;
```





代码与设计

```
assign rf_raddr1 = rj;  
assign rf_raddr2 = src_reg_is_rd ? rd :rk;  
regfile u_regfile(  
    .clk      (clk      ),  
    .raddr1   (          ),  
    .rdata1   (rj_value ),  
    .raddr2   (          ),  
    .rdata2   (rkd_value),  
    .we       (gr_we    ),  
    .waddr    (          ),  
    .wdata    (rf_wdata )  
);
```





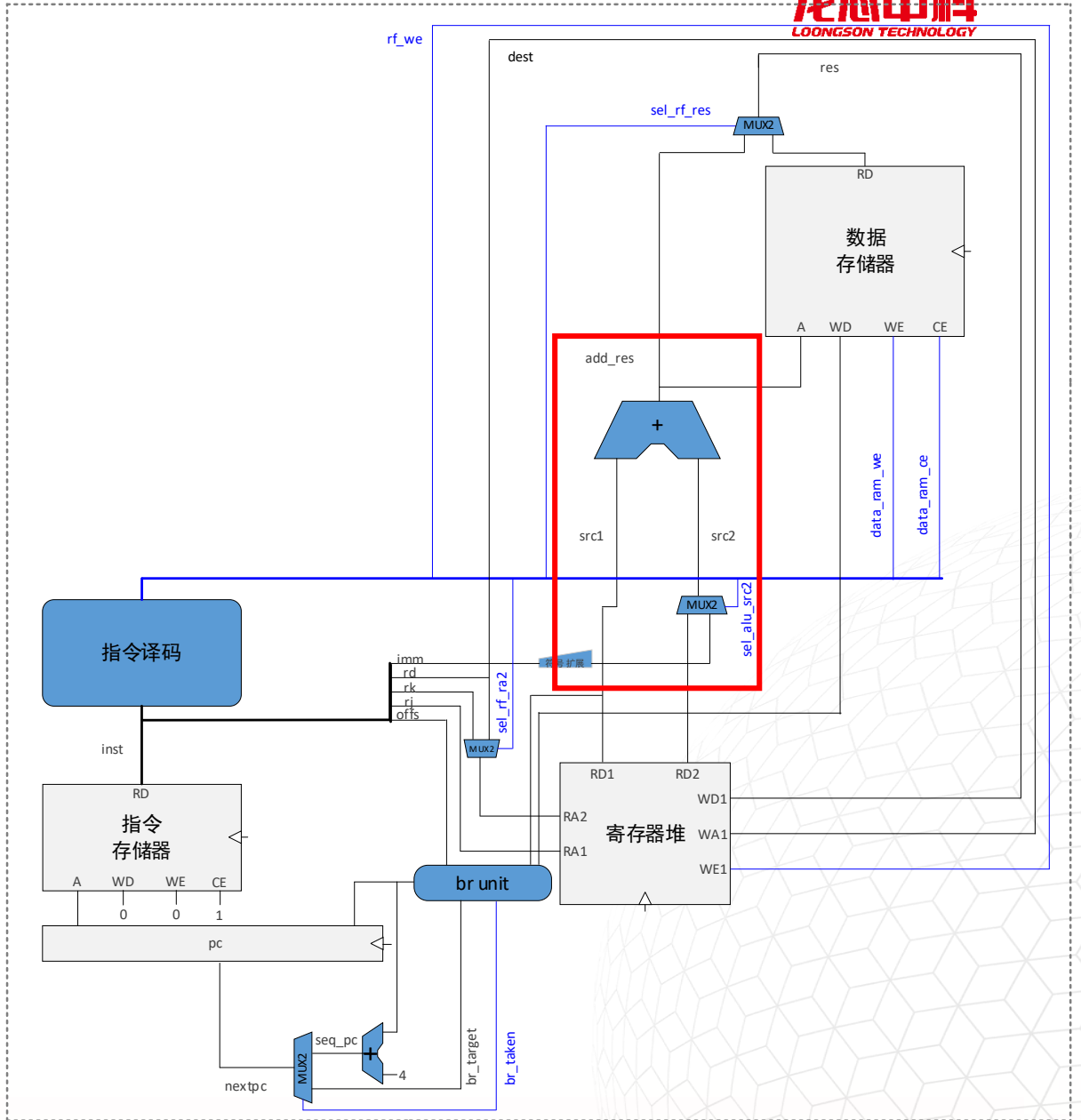
• • • • •





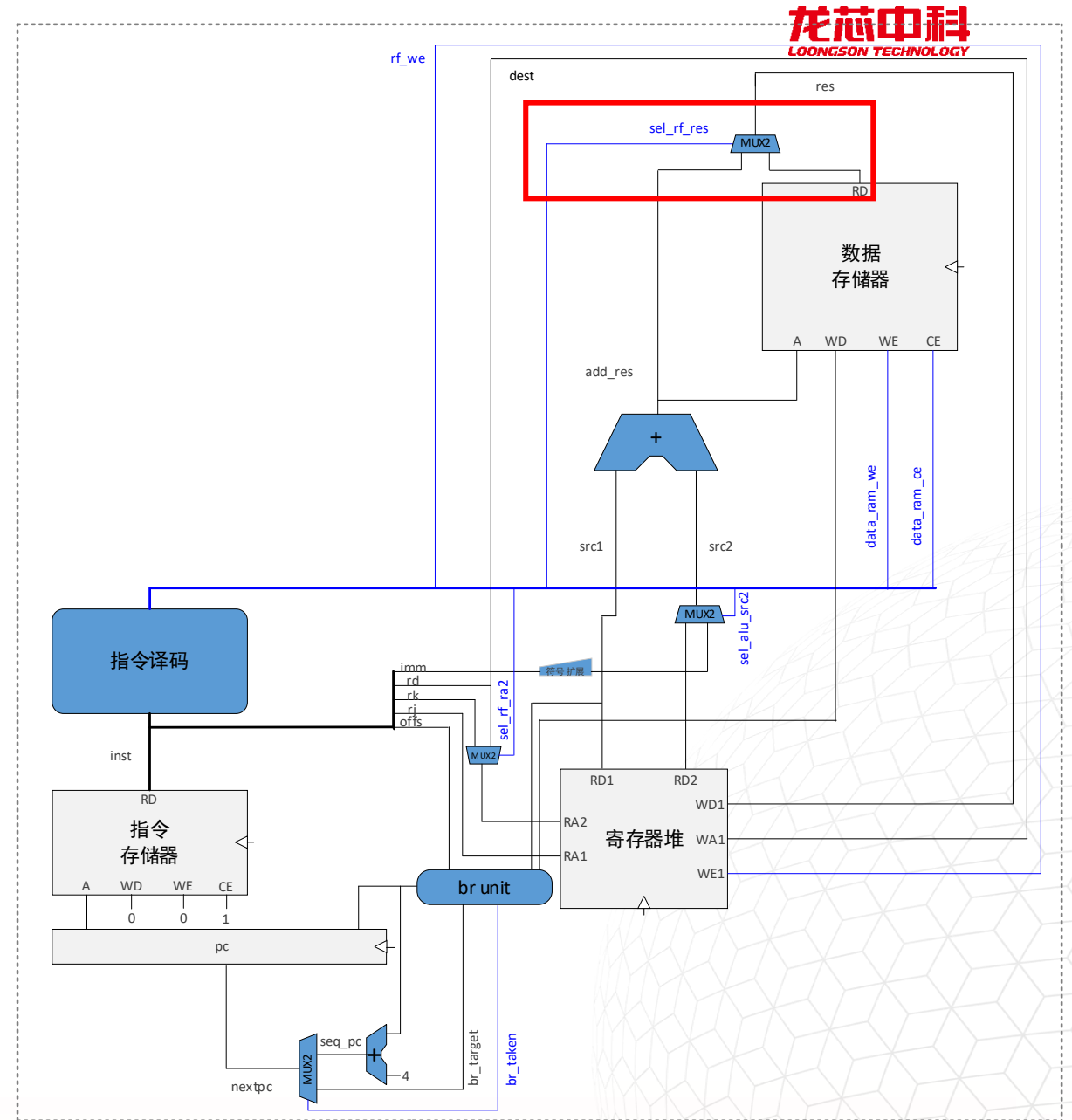
代码与设计

```
.....  
assign imm          = {{20{i12[11]}}},i12[11:0]};  
assign alu_src1      = rj_value;  
assign alu_src2      = //在这里实现alu_src2信号  
assign alu_result    = alu_src1 + alu_src2;  
.....
```





```
.....
assign rf_wdata=; //在这里完成写回寄存器值的选择
.....
```



目录

- 01 实验任务
- 02 CPU结构设计方案
- 03 参考设计代码解读
- 04 设计开发环境介绍



CPU 设计开发环境（CPU_CDE1）组织结构介绍

`| -func`

`| --inst.ram.coe`

`| -mycpu_verify/`

`| --rtl/`

`| --thinpad_top.v`

`| --myCPU/*`

`| --testbench/`

`| --mycpu_tb.sv`

`| --run_vivado/`

`| --soc_lite.xdc`

`| --mycpu_prj1/`

`| --*.xpr`

实验任务所用的功能验证测试程序。

斐波那契数程序的二进制代码。

实验者实现的CPU的验证环境。

SoC_lite设计代码目录。

SoC_lite的顶层文件。

实验者实现的CPU的RTL代码。

功能仿真验证平台。

功能仿真顶层，该模块会模拟斐波那契数程序。

Vivado工程的运行目录。

Vivado工程设计的约束文件。

创建的第一个Vivado工程，名字为mycpu_prj1。

Vivado创建的工程文件，可直接打开。



测试程序说明

```
1c000000:      addi.w      $t0,$zero,0x0      //置第1项的0
1c000004:      addi.w      $t1,$zero,0x1      //置第2项的1
1c000008:      addi.w      $s0,$zero,0x0      //循环变量i初始化为0
1c00000c:      addi.w      $s1,$zero,0x1      //循环的步长置为1
1c000010:      ld.w        $a0,$zero,1024     //读取拨码开关输入的终止值

      loop:
1c000014:      add.w       $t2,$t0,$t1        //f(i) = f(i-2) + f(i-1)
1c000018:      addi.w      $t0,$t1,0x0        //记录f(i-1)
1c00001c:      addi.w      $t1,$t2,0x0        //记录f(i)
1c000020:      add.w       $s0,$s0,$s1        //i++
1c000024:      bne        $s0,$a0,loop       //if i!=n, goto loop
1c000028:      st.w        $t2,$zero,1025     //将f(n)的值输出到数码管上

      end:
1c00002c:      bne        $s1,$zero,end       //测试完毕，进入死循环
```



为人民做龙芯