实验 7 报告

学号: 2018K8009929043 2018K8009929035
姓名: 曾冕 张翔雨
箱子号： 06

一、实验任务（10%）

本实验主要在 lab6 的前提下添加了常用的大部分转移指令和访存指令，其中添加转移指令较为容易，在添加访存指令的时候涉及到了寄存器堆和数据前递逻辑的大范围调整，比较复杂。

二、实验设计（40%）

（一）总体设计思路

本次的 CPU 在 lab6（运算类指令实现）的基础上加入了对跳转指令和访存指令的支持。转移指令的实现主要在 ID 模块，涉及到了 br_bus 的相应信号处理和例行译码，而访存指令的实现由于涉及到数据前递，非对称访存等，需要对寄存器堆，数据前递和 MEM,EXE 模块中的逻辑进行大范围调整。总体而言涉及到改动的模块有 ID 模块，EXE 模块，MEM 模块，reg_file 模块和 WB 模块。（当然总线位宽也要做相应调整）

1. ID 模块设计思路：例行对新指令进行 32 位指令分析译码生成对应信号。对于跳转指令，向外传输内容变化集中于源操作数种类的控制信号，寄存器使能信号和 pc 控制信号 br_bus 等。内部信号的变化在详细叙述中描述。对于 访存指令，将访存的信号单个列出，并在总线中向下一级流水传递。同时也将对应的控制信号更新。并考虑到字节写使能信号的变化，对于前递回来的数据进行了拆分。最后根据阻塞的逻辑对阻塞信号进行更新
2. EXE 模块设计思路：变化较少，主要对来自 ID 阶段的 store 类指令进行更新，将对应的数据用位运算的方式进行拼接，并生成 相应的 data_sram_wen 信号传递给内存
3. MEM 模块设计思路：变化较多（由于主要负责的就是 load 指令处理），主要对来自 ID 阶段流水过来的 load 类指令进行更新，将来自 data_sram_rdata 的数据用位运算的方式进行拼接，得到预备结果。同时生成写使能信号，在 WB 模块进行更新
4. Reg_file 模块设计思路：更改为字节写使能信号方式，将 32 位数据拆成 4 个部分来根据四位写使能分别读取。

最后的总体涉及思路如下图 1 所示。

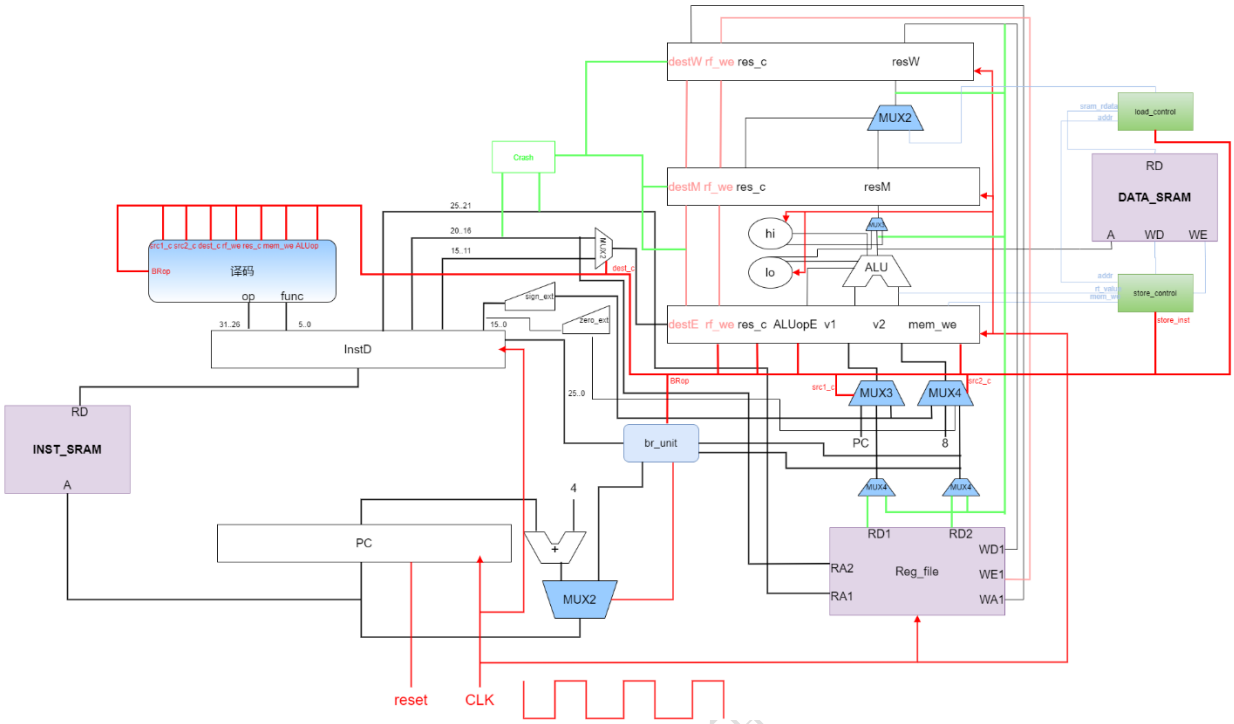

图 01: 整体数据通路设计

（二）重要模块 1 设计：ID 模块

1、工作原理

根据输入指令，通过复杂的内部信号处理输出对应的信号，控制下一步各分模块的控制逻辑。

Lab7 中，更改了 ds_to_es_bus 的位宽，传输了所有的访存信号，具体内容见功能描述，br_bus 位宽不变，但是内容变化较大

2、接口定义（仅展示 lab6 更新）

名称	方向	位宽	功能描述
ds_to_es_bus	OUT	159	到 exe_stage 的数据（包含所有的访存信号（1bit 形式列出））
Br_bus	OUT	33	到 IF 段的 pc 信号更新

3、功能描述

总线修改的信号如下图 2 所示：

```
assign ds_to_es_bus = {inst_sw     ,  //158:158
                       inst_lb     ,  //157:157
                       inst_lbu    ,  //156:156
                       inst_lh     ,  //155:155
                       inst_lhu    ,  //154:154
                       inst_lwl    ,  //153:153
                       inst_lwr    ,  //152:152
                       inst_sb     ,  //151:151
                       inst_sh     ,  //150:150
                       inst_swl    ,  //149:149
                       inst_swr    ,  //148:148
```

图 02：总线中新增的信号

将所有访存信号以 1 位位宽的形式列出，传到下一级流水

（译码部分略去）

对 aluop[0] 进行了更新（添加了对应的访存和跳转到加法信号内）

对下一级流水的控制信号的更新如下图 3 所示：

```
assign src1_is_pc    = inst_jal | inst_bltzal | inst_bgezal | inst_jalr;
assign src2_is_simm = inst_addiu | inst_lui | inst_lw | inst_sw | inst_addi | inst_slti | inst_sltiu | inst_lb | inst_lbu
                    | inst_lh | inst_lhu | inst_lwl | inst_lwr | inst_sb | inst_sh | inst_swl | inst_swr ;
assign src2_is_zimm = inst_ori | inst_xori |inst_andi;
assign src2_is_8     = inst_jal | inst_bltzal | inst_bgezal | inst_jalr;
assign dst_is_r31    = inst_jal | inst_bgezal | inst_bltzal;
assign dst_is_rt     = inst_addiu | inst_lui | inst_lw | inst_addi | inst_slti | inst_sltiu | inst_andi | inst_ori
                    | inst_xori | inst_lb | inst_lbu | inst_lh | inst_lhu | inst_lwl | inst_lwr;
assign gr_we         = ~inst_sw & ~inst_beq & ~inst_bne & ~inst_jr & ~inst_mthi & ~inst_mtlo & ~inst_bgez
                     & ~inst_bgtz & ~inst_blez & ~inst_bltz & ~inst_j & ~inst_lwl & ~inst_lwr & ~inst_sb
                     & ~inst_sh & ~inst_swl & ~inst_swr ;
assign ds_gr_we        = {4{gr_we}};
assign mem_we        = inst_sw | inst_sb | inst_sh | inst_swl | inst_swr;
```

图 03：总线中修改的信号

对源操作数的分类进行了更新，写使能信号进行了更新，同时将使能信号改为字节写使能形式。

对阻塞信号的控制更新如下图 4 和下图 5 所示所示：

```
assign crash_on_es_valid = (es_to_ms_bus[72] || es_to_ms_bus[71] || es_to_ms_bus[70] || es_to_ms_bus[69]) && es_to_ms_valid;
assign crash_on_ms_valid = (ms_to_ws_bus[72] || ms_to_ws_bus[71] || ms_to_ws_bus[70] || ms_to_ws_bus[69]) && ms_to_ws_valid;
assign crash_on_ws_valid = (rf_we[3] || rf_we[2] || rf_we[1] || rf_we[0]) && ws_valid;

wire crash_raddr1;
wire crash_raddr2;

assign crash_raddr1 =   (rf_raddr1 == 5'b0 ) ? 1'b0 :
                        ((rf_raddr1 == es_to_ms_bus[68:64]) && crash_on_es_valid) ? 1'b1 :
```

图 04：对阻塞有效来源判断的修改

利用字节写使能的形式，只要有一位有效就可以进入判断，如果地址相同，则说明这个地址发生了数据相关阻塞。

```
assign crash   = (inst_addu|inst_add|inst_subu|inst_sltu|inst_slt|inst_and|inst_or|inst_xor|inst_nor
                |inst_bne|inst_beq|inst_sw|inst_sb|inst_sh|inst_swl|inst_swr|inst_sllv|inst_srlv|inst_srav) ?
                (crash_raddr1 | crash_raddr2) :
                (inst_addiu|inst_jr|inst_lw|inst_addi| inst_slti | inst_sltiu | inst_ori | inst_xori | inst_andi
                | inst_mtlo | inst_mthi | inst_lb | inst_lbu | inst_lh | inst_lhu | inst_lwl | inst_lwr
                | inst_bgez | inst_bgtz | inst_blez | inst_bltz | inst_bgezal | inst_bltzal) ?
                crash_raddr1 :
                (src1_is_sa) ? crash_raddr2 :
                1'b0;
```

图 05：对阻塞信号的修改

根据相应的指令相应的源操作数的地址和 1 有关还是 2 有关来对阻塞信号进行修改。

更改为位拼接实现方式后，需要对前递的数据进行拆分，如下图 6 的代码截图所示：

```
rs_value[ 7: 0] =    (rf_raddr1 == 5'b0 ) ? 8'h00 :
                     ((rf_raddr1 == es_to_ms_bus[68:64]) && es_to_ms_bus[69] && crash_on_es_valid) ? es_forward_data [ 7: 0] :
                     ((rf_raddr1 == ms_to_ws_bus[68:64]) && ms_to_ws_bus[69] && crash_on_ms_valid) ? ms_forward_data [ 7: 0] :
                     ((rf_raddr1 == rf_waddr) && rf_we[0] && crash_on_ws_valid) ?  rf_wdata[ 7: 0]  :
                     rf_rdata1[ 7: 0];
rt_value[ 7: 0] =    (rf_raddr2 == 5'b0 ) ? 8'h00 :
                     ((rf_raddr2 == es_to_ms_bus[68:64]) && es_to_ms_bus[69] && crash_on_es_valid) ? es_forward_data [ 7: 0] :
                     ((rf_raddr2 == ms_to_ws_bus[68:64]) && ms_to_ws_bus[69] && crash_on_ms_valid) ? ms_forward_data [ 7: 0]:
                     ((rf_raddr2 == rf_waddr) && rf_we[0] && crash_on_ws_valid) ?  rf_wdata[ 7: 0]  :
                     rf_rdata2[ 7: 0];
```

图 06：对前递数据的拆分（仅展示前 8 位，剩下 24 位高度相似，故在此略去）

最后，对于转移类指令更新到 IF 阶段的 br_bus 信号，作出的更新如下图 7 所示：

```
assign rs_eq_rt = (rs_value == rt_value);

wire rs_bltz;
wire rs_bgtz;

assign rs_bltz = (rs_value[31] == 1'b1);
assign rs_bgtz = (rs_value[31] == 1'b0) && (rs_value != 32'd0);

assign br_taken = (   inst_beq  &&  rs_eq_rt
                   || inst_bne  && !rs_eq_rt
                   || inst_bgez && !rs_bltz
                   || inst_bgtz &&  rs_bgtz
                   || inst_bltz &&  rs_bltz
                   || inst_blez && !rs_bgtz
                   || inst_j
                   || inst_jalr
                   || inst_bgezal && !rs_bltz
                   || inst_bltzal && rs_bltz
                   || inst_jal
                   || inst_jr
                  ) && ds_valid;
assign br_target = (inst_beq || inst_bne || inst_bgez || inst_bgtz || inst_bltz || inst_blez
                    || inst_bltzal || inst_bgezal) ? (fs_pc + {{14{imm[15]}}, imm[15:0], 2'b0}) :
                   (inst_jr || inst_jalr)              ? rs_value :
                   /*inst_jal , j*/         {fs_pc[31:28], jidx[25:0], 2'b0} ;
```

图 07：br_bus 部分的更新内容

Br_taken 这个 1 位信号根据新加入指令的要求添加了是否跳转的逻辑判断

Br_target 根据是否连接寄存器等对于加入的跳转指令进行了更新。

（三）重要模块 2 设计：EXE 模块（仅描述 lab7 更新）

1、工作原理

Lab7 中，exe 根据 ID 阶段流水过来的 STORE 类信号进行了数据存入 data_sram 的预处理工作

2、接口定义

名称	方向	位宽	功能描述
ds_to_es_bus	IN	159	来自 ID 的数据（包含所有的访存信号（1bit 形式列出））

3、功能描述

根据从 ID 阶段流水过来的 store 类信号和 es_rt_value，对要写入 data_sram 的数据进行处理，并生成对应的写入 4 位使能信号。

数据的拼接如下图 8 所示：

```
assign addr_low = data_sram_addr[1:0];
assign swl_data = {32{(addr_low==2'b00)}}&{24'h000000,es_rt_value[31:24]}|
                  {32{(addr_low==2'b01)}}&{16'h0000,es_rt_value[31:16]}|
                  {32{(addr_low==2'b10)}}&{8'h00,es_rt_value[31:8]}|
                  {32{(addr_low==2'b11)}}&es_rt_value;
assign swr_data = {32{(addr_low==2'b00)}}&es_rt_value|
                  {32{(addr_low==2'b01)}}&{es_rt_value[23:0],8'h00}|
                  {32{(addr_low==2'b10)}}&{es_rt_value[15:0],16'h00}|
                  {32{(addr_low==2'b11)}}&{es_rt_value[7:0],24'h000000};

assign st_data  = inst_sb ? {4{es_rt_value[ 7:0]}} :
                  inst_sh ? {2{es_rt_value[15:0]}} :
                  inst_swl?swl_data :
                  inst_swr?swr_data : es_rt_value ;
```

图 08 ： 写入数据的位拼接结果

写入 data_sram 的 4 位使能信号生成方式如下图 9 所示：

```
assign data_sram_wen   = es_mem_we&&es_valid ? ({4{inst_sb&&(addr_low==2'b00)}}&4'b0001|
                                                {4{inst_sb&&(addr_low==2'b01)}}&4'b0010|
                                                {4{inst_sb&&(addr_low==2'b10)}}&4'b0100|
                                                {4{inst_sb&&(addr_low==2'b11)}}&4'b1000|
                                                {4{inst_sh&&(addr_low==2'b10)}}&4'b1100|
                                                {4{inst_sh&&(addr_low==2'b00)}}&4'b0011|
                                                {4{inst_sw                   }}&4'b1111|
                                                {4{inst_swl&&(addr_low==2'b00)}}&4'b0001|
                                                {4{inst_swl&&(addr_low==2'b01)}}&4'b0011|
                                                {4{inst_swl&&(addr_low==2'b10)}}&4'b0111|
                                                {4{inst_swl&&(addr_low==2'b11)}}&4'b1111|
                                                {4{inst_swr&&(addr_low==2'b00)}}&4'b1111|
                                                {4{inst_swr&&(addr_low==2'b01)}}&4'b1110|
                                                {4{inst_swr&&(addr_low==2'b10)}}&4'b1100|
                                                {4{inst_swr&&(addr_low==2'b11)}}&4'b1000)
                                                :4'h0;
```

图 09：4 位写使能信号利用位运算生成的代码

在这里用了位运算的方式生成写使能信号，而不是选择器，减少硬件开销

（四）重要模块 3 设计：MEM 模块

1、工作原理

根据 EXE 阶段流水过来的总线数据，拆分出对应的 load 指令信号，完成数据的处理和寄存器堆写使能信号的生成

2、接口定义（仅展示 lab7 更新）

名称	方向	位宽	功能描述
es_to_ms_bus	IN	80	EXE 阶段流水过来的数据，包括 load 信号和 alu 结果等

3、功能描述

数据位拼接结果如下图 10 所示，大致处理方式同 MEM 阶段，但多了几条特殊 load 指令的数据处理：

```
assign lbu_data = (addr_low==2'b00)?{24'h000000,data_sram_rdata[7:0]}:
                  (addr_low==2'b01)?{24'h000000,data_sram_rdata[15:8]}:
                  (addr_low==2'b10)?{24'h000000,data_sram_rdata[23:16]}:
                  (addr_low==2'b11)?{24'h000000,data_sram_rdata[31:24]}:
                  0;
assign lb_data  = (addr_low==2'b00)?{{24{data_sram_rdata[7]}},data_sram_rdata[7:0]}:
                  (addr_low==2'b01)?{{24{data_sram_rdata[15]}},data_sram_rdata[15:8]}:
                  (addr_low==2'b10)?{{24{data_sram_rdata[23]}},data_sram_rdata[23:16]}:
                  (addr_low==2'b11)?{{24{data_sram_rdata[31]}},data_sram_rdata[31:24]}:
                  0;
assign lhu_data = (addr_low==2'b10)?{16'h0000,data_sram_rdata[31:16]}:
                  {16'h0000,data_sram_rdata[15:0]};
assign lh_data  = (addr_low==2'b10)?{{16{data_sram_rdata[31]}},data_sram_rdata[31:16]}:
                  {{16{data_sram_rdata[15]}},data_sram_rdata[15:0]};
assign lwl_data = (addr_low==2'b00)?{data_sram_rdata[7:0],24'd0}:
                  (addr_low==2'b01)?{data_sram_rdata[15:0],16'd0}:
                  (addr_low==2'b10)?{data_sram_rdata[23:0],8'd0}:
                  data_sram_rdata;
assign lwr_data = (addr_low==2'b11)?{24'd0,data_sram_rdata[31:24]}:
                  (addr_low==2'b10)?{16'd0,data_sram_rdata[31:16]}:
                  (addr_low==2'b01)?{8'd0,data_sram_rdata[31:8]}:
                  data_sram_rdata;
```

图 10：load 部分从 sram 读取数据的处理

写使能信号的处理如下图 11 所示：

```
assign lwl_wen  = (addr_low==2'b00)?4'b1000:
                  (addr_low==2'b01)?4'b1100:
                  (addr_low==2'b10)?4'b1110:
                  4'b1111;
assign lwr_wen  = (addr_low==2'b00)?4'b1111:
                  (addr_low==2'b01)?4'b0111:
                  (addr_low==2'b10)?4'b0011:
                  4'b0001;

assign ms_final_gr_we = inst_lwl ? lwl_wen :
                        inst_lwr ? lwr_wen :
                        ms_gr_we;
```

图 11：MEM 阶段流水至 WB 阶段写使能信号的处理

在这里对于写使能信号当 LWL 和 LWR 信号的时候使用了 LWL 和 LWR 自身的写使能信号，避免在 WB 阶段出现数据写入错误情况

Mem_result 相应的选择信号更改略去

（五）重要模块 4 设计：reg_file 模块

1、工作原理

将 1 位写使能信号更改为 4 位字节写使能信号驱动

2、接口定义（仅展示 lab7 更新）

名称	方向	位宽	功能描述
we	IN	4	字节写使能信号

3、功能描述

按照讲义，更改为 4 位字节写使能驱动：如下图 12 所示

```
always @(posedge clk) begin
    if (we[0] || we[1] || we[2] || we[3])
    begin
        rf[waddr][ 7: 0]  <= we[0] ? wdata[ 7: 0] : rf[waddr][ 7: 0];
        rf[waddr][15: 8]  <= we[1] ? wdata[15: 8] : rf[waddr][15: 8];
        rf[waddr][23:16]  <= we[2] ? wdata[23:16] : rf[waddr][23:16];
        rf[waddr][31:24]  <= we[3] ? wdata[31:24] : rf[waddr][31:24];
    end
end
```

图 12：寄存器写使能信号的处理

三、实验过程（50%）

（一）实验流水账

10.20 晚上大约 2 小时　小组成员完成了指令译码和阅读讲义

10.21 晚上　　小组成员用约 1 小时完成了数据通路的大改造（4 位写使能），再用两小时添加指令并成功仿真通过

10.24 晚上　　开始写实验报告，大约 3 小时半完成

（二）错误记录

1、错误 1：位宽错误

修改了数据位宽却没有更改代码中直接从 bus 中取值的位出错，常犯的错误，所以在此简单的记录一下，除了这个以外没有什么大的，需要对比波形才能找出来的错误了。

2、错误 2：多余阻塞

（1）错误现象

无（仿真时间比应有的长）

（2）分析定位过程

在某次浏览波形的时候发现出现了意料之外的阻塞，遂排查阻塞信号的赋值逻辑

（3）错误原因

在进行阻塞判断时复用了 Lab4 中阻塞信号，这会对 EXE,MEM,WB 三个模块正在进行的指令进行判断，事实上只需要判断 EXE 阶段是否为 LOAD 指令即可，当时 Lab5 时也想到了这一点，于是在 crash 信号对 EXE 状态有效信号以及 EXE 状态的 load_op 进行了与操作以确保阻塞是在 EXE 阶段 load 指令时才产生的，事实上这是非常错误的一点，因为很可能是后两个阶段发生操作数冲突而 EXE 阶段是 load 指令就会阻塞，这点在 Lab7　load 指令大量增加后才比较明显。

（4）修正效果

注释掉对后两个阶段的判断，仿真时间大幅度缩短。

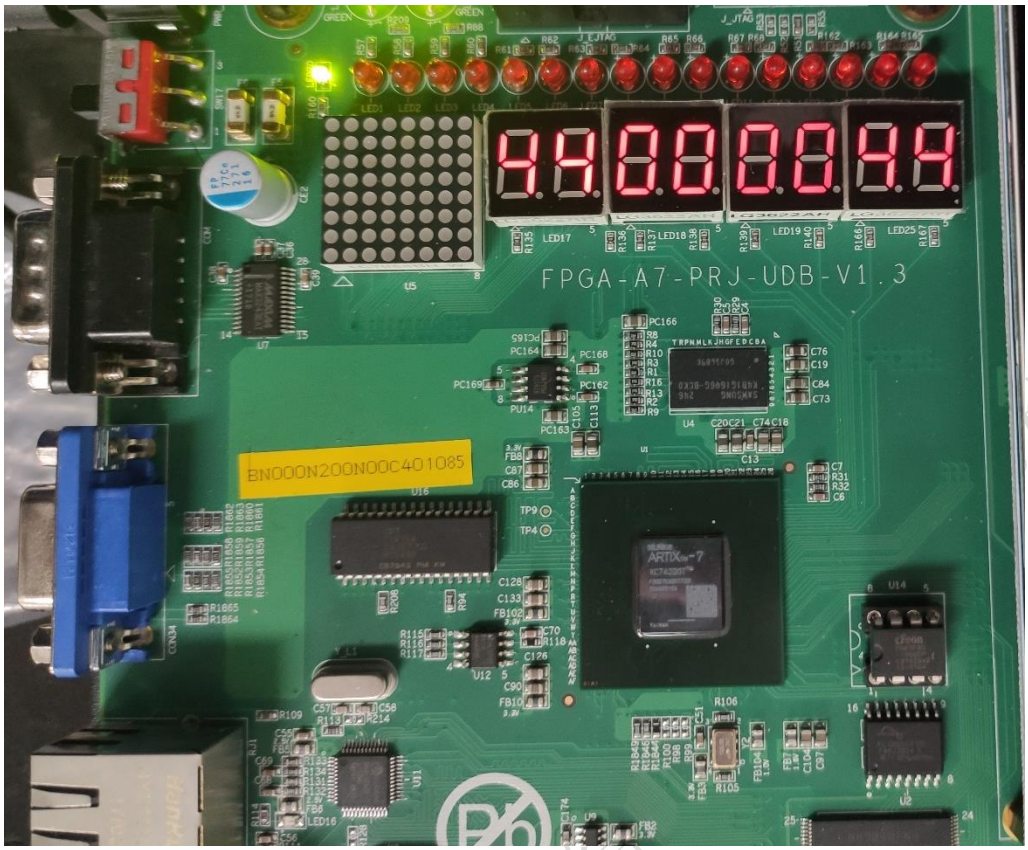

图 13 上板通过照片

四、实验总结（可选）

本次实验完成的跳转指令和访存指令主要集中在 ID 阶段的处理。在对相应的传输至 EXE 阶段的控制信号进行修改和进行访存操作的时候需要对指令较为熟悉，得益于上学期计组的磨练，这部分完成的较为轻松。

同时 lab7 较 lab6 一个较大的改动就是对寄存器堆进行了大改动，从 1 位写使能变成了 4 位写使能信号，因为这个改动也导致前递的数据通路的逻辑需要大修改，这一部分比较繁琐。最后的测试文件中并没有测试 LWR 和 LWL 连续出现的情况下的前递处理，也可能是 MIPS 在编译时进行了处理，导致两者之间会有空指令（因为指令手册上的描述就是先从寄存器堆读值再写入故这样猜测），这样讲义中的方案 1 和方案 2 在仿真时间上应该是是一样的，没有体现出差别，这一点较为可惜。