# 实验 3 报告

学号 2018K8009929021
姓名 袁欣怡
箱子号 10

## 一、实验任务（10%）

学习了解无阻塞五级流水线 CPU 设计思路，并调试有错误的 CPU 设计代码。本次实验包括两个子任务：

子任务一：阅读代码，画出流水线 CPU 的设计结果框图。

子任务二：结合仿真结果和上板测试，对代码进行调试，找出其中的 7 处错误。

## 二、实验设计（40%）

### (一) 总体设计思路

缺少细节（参考第三讲ppt，图附在最后）
一般一个框表示一个逻辑部件，而不是一个stage

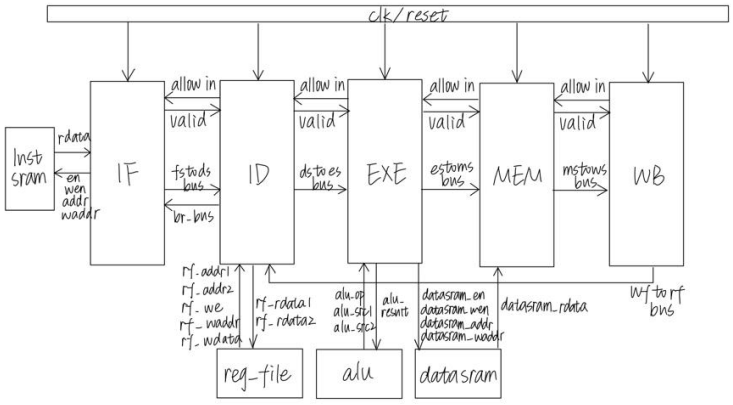



图 1 硬件结构设计图

代码设计中，主要包括了七个模块，包括 ALU、寄存器堆、取指模块、译码模块、执行模块、访存模块、写回模块。该设计还使用了两个 IP 核，Inst_RAM 和 Data_RAM 核。

每当时钟信号到来的时候，IF 模块负责更新 PC 值，从 inst_sram 中取指令，并把指令和当前 PC 值放到 fs_to_ds_bus 总线上。同时，IF 模块还会将 fs_to_ds_valid 信号拉高，目的是告诉 ID 模块数据已经准备好，等到 ID 接收数据时即可完成数据传递。

ID 模块会对接收到的指令进行译码，并判断指令的类型。如果有分支指令，则需要判断是否需要跳转，否则就将译码结果全部送到 ds_to_es_bus 总线上，并把 ds_to_es_valid 拉高，告诉 EXE 模块数据已经准备好。此外，ID 模块还需要接受来自 WB 模块的写回寄存器的数据，在译码的同时调用 reg_file 模块，写回数据。

EXE 模块接收到 ID 的数据后，根据指令调用 ALU 进行计算。同时 EXE 还需要判断该指令是否需要写内存（e.g. sw 指令），并调整内存写使能信号 data_sram_wen。EXE 模块需要将计算结果等输送到 es_to_ms_bus 总线。并把 es_to_ms_valid 拉高，等到 MEM 模块接收数据，同时给 data_sram 发送读内存的信号。

MEM 模块，数据传入 MEM 模块，同时接受 data_sram 传来的数据。MEM 需要判断最终写入内存的数据是哪个，EXE 模块从 ALU 中得到的结果还是 data_sram 发来的数据。确定后 MEM 模块将这个数据传到 ms_to_ws_bus 总线上，并把 ms_to_ws_valid 拉高，等待 WB 模块读数据。

WB 模块允许数据进入后，需要根据传进来的数据，判断是否要写寄存器，同时确认写寄存器的地址。WB 模块需要将数据传送到 ws_to_rf_bus 总线上，让 ID 模块接收。

## (二) 重要模块 1：IF 模块

### 1. 工作原理

从 inst_ram 中读取指令，处理 PC 的值，并将指令传递给 bus，下一周期再传递给 ID 模块。

### 2. 接口定义

表 1 IF 模块的接口定义

| 名称 | 方向 | 位宽 | 功能描述 |
|---|---|---|---|
| ds_allowin | IN | 1 | IDstage 流水线接收允许信号，高电平有效 |
| br_bus | IN | 33 | 分支指令总线，包括是否有分支指令跳转信号(1 位)、分支指 令跳转目标(32 位) |
| fs_to_ds_valid | OUT | 1 | IFstage 到 IDstage 是否有效，高电平有效 |
| fs_to_ds_bus | OUT | 64 | IFstage 到 IDstage 总线，包含从 sram 取的指令(高 32 位)、以 及 IFstage 的当前 PC |
| inst_sram_en | OUT | 1 | 向 sram 中取指使能，高电平有效 |
| inst_sram_wen | OUT | 4 | 向 sram 中取指写使能 |
| inst_sram_addr | OUT | 32 | IF 过程向 sram 读写数据的地址 |
| inst_sram_wdata | OUT | 32 | IF 过程向 sram 写的数据 |
| inst_sram_rdata | IN | 32 | IF 过程读出的指令数据 |

### 3. 功能描述

从 sram 取出当前 PC 地址存储的指令。

## (三) 重要模块 2：ID 模块

### 1. 工作原理

对指令进行译码，并且判断是否需要跳转。同时接收 WB 模块的数据，传递给寄存器堆。

### 2. 接口定义

表 2 ID 模块的接口定义

| 名称 | 方向 | 位宽 | 功能描述 |
|---|---|---|---|
| es_allowin | IN | 1 | EXEstage 流水线接收允许信号，高电平有效 |

| | | | |
|---|---|---|---|
| ds_allowin | OUT | 1 | IDstage 流水线接收允许信号，高电平有效 |
| fs_to_ds_valid | IN | 1 | IFstage 到 IDstage 是否有效，高电平有效 |
| fs_to_ds_bus | IN | 64 | IFstage 到 IDstage 总线，包含从 sram 取的指令(高 32 位)、以 及 IFstage 的当前 PC |
| ds_to_es_valid | OUT | 1 | IDstage 到 EXEstage 方向有效信号，高电平有效 |
| ds_to_es_bus | OUT | 136 | IDstage 到 EXEstage 数据总线，包含 alu 操作信号(12 位)、从 内存加载数信号(1 位)、操作数 1 的来源(2 位)、操作数 2 的来源(2 位)、寄存器堆写使能(1 位)、内存写使能(1 位)、寄存器写地址(5 位)、立即数(16 位)、rs 寄存器的值 (32 位)、rt 寄存器的值(32 位)、IDstage 的当前 PC(32 位) |
| br_bus | OUT | 33 | 分支指令总线，包括是否有分支指令跳转信号(1 位)、分支指 令跳转目标(32 位) |
| ws_to_rf_bus | IN | 38 | 写寄存器总线，包含寄存器写使能(1 位)、寄存器写地址(5 位)、寄存器写数据(32 位) |

### 3. 功能描述

对 IF 传来的指令进行译码，大致确定指令类型（是否需要加载，是否需要跳转等）。在译码的数据和控制信号在下一时钟周期传递给 EXE 模块，并接受 WB 模块返回的数据。

## (四) 重要模块 3 :EXE 模块

### 1. 工作原理

当 EXEstage 流水线接收允许信号为高电平且向 EXEstage 流水线方向有效(ds_to_es_valid)时，调用 ALU 模 块进行计算。

### 2. 接口定义

表 3 EXE 模块的接口定义

| 名称 | 方向 | 位宽 | 功能描述 |
|---|---|---|---|
| ms_allowin | IN | 1 | MEMstage 流水线接收允许信号，高电平有效 |
| es_allowin | OUT | 1 | EXEstage 流水线接收允许信号，高电平有效 |
| ds_to_es_valid | IN | 1 | IDstage 到 EXEstage 方向有效信号，高电平有效 |
| ds_to_es_bus | IN | 136 | IDstage 到 EXEstage 数据总线，包含 alu 操作信号(12 位)、从 内存加载数信号(1 位)、操作数 1 的来源(2 位)、操作数 2 的来源(2 位)、寄存器堆写使能(1 位)、内存写使能(1 位)、寄存器写地址(5 位)、立即数(16 位)、rs 寄存器的值(32 位)、rt 寄存器的值(32 位)、IDstage 的当前 PC(32 位) |
| es_to_ms_valid | OUT | 1 | EXEstage 到 MEMstage 是否有效，高电平有效 |
| es_to_ms_bus | OUT | 71 | EXEstage 到 MEMstage 总线，包含是否从内存加载数至寄存器 信号(1 位)、寄存器写使能(1 位)、寄存器写地址(5 位)、ALU 计算结果(32 位)、EXEstage 的当前 PC(32 位) |
| data_sram_en | OUT | 1 | 向 sram 中读数据使能，高电平有效 |
| date_sram_wen | OUT | 4 | 向 sram 中写数据使能，高电平有效 |
| data_sram_addr | OUT | 32 | 向 sram 中读写数据的地址，地址为 ALU 计算的结果 |
| data_sram_rdata | OUT | 32 | 从 sram 中读取的数据 |

### 3. 功能描述

根据 ID 模块得到的不同的指令译码执行不同的指令，调用 ALU。需要注意的是 sw 指令也在该阶段完成。

## (五) 重要模块 4 设计:MEM 模块

### 1. 工作原理

当 MEM 模块的允许接收，且向 MEM 模块的流水线有效时，向 WB 模块输出写回阶段使用的数据和地址等。

### 2. 接口定义

表 4 MEM 模块的接口定义

| 名称 | 方向 | 位宽 | 功能描述 |
|---|---|---|---|
| ws_allowin | IN | 1 | MEMstage 流水线接收允许信号，高电平有效 |
| ms_allowin | OUT | 1 | EXEstage 流水线接收允许信号，高电平有效 |
| es_to_ms_valid | IN | 1 | EXEstage 到 MEMstage 是否有效，高电平有效 |
| es_to_ms_bus | IN | 71 | EXEstage 到 MEMstage 总线，包含是否从内存加载数至寄存器 信号(1 位)、寄存器写使能(1 位)、寄存器写地址(5 位)、ALU 计算结果(32 位)、EXEstage 的当前 PC(32 位) |
| ms_to_ws_valid | OUT | 1 | MEMstage 到 WBstage 方向有效信号，高电平有效 |
| ms_to_ws_bus | OUT | 70 | MEMstage 到 WBstage 数据总线，包含寄存器写使能(1 位)、 寄存器写地址(5 位)、最终传输的数据(从 ALU 来的或从内 存中来的，取决于 es_to_ms_bus 中的是否从内存加载数至寄存 器的信号，32 位)、MEMstage 的当前 PC(32 位) |
| data_sram_rdata | IN | 32 | 从 sram 中读取的数据 |

### 3. 功能描述

向 WB 模块传输可能要写回寄存器的数据，它可能是来自于内存，也可能来自于 EXE 模块计算出的 alu_result，这需要根据 ms_res_from_mem 信号来判断。

## (六) 重要模块 5 :WB 模块

### 1. 工作原理

当 WBstage 流水线接收允许信号为高电平且向 WBstage 流水线方向有效(ms_to_ws_valid)时，向给定地址的寄存器写回数据。

### 2. 接口定义

表 5 WB 模块的接口定义

| 名称 | 方向 | 位宽 | 功能描述 |
|---|---|---|---|

| | | | |
|---|---|---|---|
| ws_allowin | OUT | 1 | MEMstage 流水线接收允许信号，高电平有效 |
| ms_to_ws_valid | IN | 1 | MEMstage 到 WBstage 方向有效信号，高电平有效 |
| ms_to_ws_bus | IN | 70 | MEMstage 到 WBstage 数据总线，包含寄存器写使能(1 位)、 寄存器写地址(5 位)、最终传输的数据(从 ALU 来的或从内 存中来的，取决于 es_to_ms_bus 中的是否从内存加载数至寄存 器的信号，32 位)、MEMstage 的当前 PC(32 位) |
| ws_to_rf_bus | OUT | 38 | 写寄存器总线，包含寄存器写使能(1 位)、寄存器写地址(5 位)、寄存器写数据(32 位) |
| debug_wb_pc | OUT | 32 | WBstage 的 PC 值，用于 debug |
| debug_wb_rf_wen | OUT | 4 | WBstage 寄存器堆写使能，用于 debug |
| debug_wb_rf_wnum | OUT | 5 | WBstage 写寄存器堆地址，用于 debug |
| debug_wb_rf_wdata | OUT | 32 | WBstage 写寄存器堆数据，用于 debug |

3. **功能描述**

向寄存器写回数据，数据来自 MEM 模块到 WB 模块的数据总线。

# 三、实验过程（50%）

## (一) 实验流水账

2020.9.27 16:00-18:00 阅读讲义，完成任务一

2020.9.27 18:30-22:30 进行任务二

2020.9.28 20:00-22:00 完成任务二和实验报告

## (二) 错误记录

### 1. 错误 1：信号为 X

**（1） 错误现象**

信号 ds_valid 为 X，fs_to_ds_valid 等几个 valid 信号均为 X。

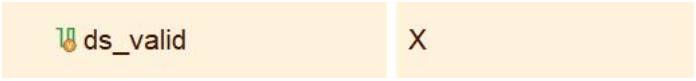



图 2 信号 ds_valid

**（2） 分析定位过程**

由于几个 valid 信号均为 X 值，考虑到流水线层层递进的关系，确定为 ds_to_es_valid 信号赋值时出现问题。在源码中 ID 模块中，从给 ds_to_es_valid 的赋值语句中可以判定时 ds_valid 的值出现问题，因此发现没有给它赋值。

**（3） 错误原因**

变量 ds_valid 没有赋值。

（4） 修正效果

可以仿照其他模块给 ds_valid 信号赋值。赋值成功后信号都恢复正常。

## 2. 错误 2：信号为 Z

（1） 错误现象

信号 load_op 没有赋初值，因此显示为 Z。

（2） 分析定位过程

发现 ds_to_es_bus 中间一段值出错。翻看源码时，将鼠标放在变量上来查看变量的值，于是发现 load_op 的值为 Z。

```
assign ds_to_es_bus = {alu_op      ,  //135:124
                       load_op     ,  //123:123
                       src1_is_sa  ,  //122:122
                       src1_is_pc  ,  //121:121
                       src2_is_imm ,  //120:120
                       src2_is_8   ,  //119:119
                       gr_we       ,  //118:118
                       mem_we      ,  //117:117
                       dest        ,  //116:112
                       imm         ,  //111:96
                       rs_value    ,  //95 :64
                       rt_value    ,  //63 :32
                       ds_pc          //31 :0
                      };
```

图 3 ds_to_es_bus 赋值语句

（3） 错误原因

变量 load_op 没有赋值。

（4） 修正效果

添加给 load_op 赋值的语句 assign load_op = inst_lw。修改之后 load_op 恢复正常。

## 3. 错误 3:信号为 Z

（1） 错误现象

变量 op_d 和 func_d 的值最高位均为 Z。

（2） 分析定位过程

这两个信号由 decoder_6_64 负责译码，这个模块在 tools.v 中。译码部分使用了循环语句，而且只有最高位出错，所以怀疑循环语句的判断条件错误。

（3） 错误原因

循环语句 generate for 中，变量 i 的变化范围为 0 至 62，没有包括最高位。

（4） 修正效果

修改循环条件，使 i 变化范围为 0 到 63。

```
genvar i;
generate for (i=0; i<64; i=i+1) begin : gen_for_dec_6_64
    assign out[i] = (in == i);
end endgenerate
```

图 4 修改后的循环条件

修改后变量赋值成功。

## 4. 错误 4：ALU 模块调用错误

### （1） 错误现象

控制台报错，提示 wb_rf_wdata 出错。

```
reference: PC = 0xbfc00000, wb_rf_wnum = 0x08, wb_rf_wdata = 0xffffffff
mycpu    : PC = 0xbfc00000, wb_rf_wnum = 0x08, wb_rf_wdata = 0xfffffffe
```

图 5 控制台报错信息

### （2） 分析定位过程

经过追溯发现出错的数据为 alu_result。查看调用 ALU 模块的部分时，发现 ALU 模块的输入 alu_src1 与错误的端口 es_alu_src2 相连。

### （3） 错误原因

模块 ALU 调用时连接到了错误的端口。

### （4） 修正效果

修改连接的端口，修改后同一指令处不再报错。

## 5. 错误 5:组合环

### （1） 错误现象

控制台在 PC=0xbfc00000 时报错（报错信息见图 5），在检查 ALU 模块时，发现 or_result 和 alu_result 之间形成了组合环。

### （2） 分析定位过程

检查 ALU 模块的源码时发现其中的 or_result 和 alu_resut 之间形成了环。

### （3） 错误原因

变量 or_result 的赋值语句出错。

### （4） 修正效果

因为 or_result 只和 alu_src1 和 alu_src2 有关，因此将 or_result 的赋值语句中的 alu_result 删除。

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

图 6 修改 or_result 的赋值语句

修改后同一指令处不再报错。

## 6. 错误 6:跳转指令 beq 出错

### （1） 错误现象

控制台报错，提示 PC 出错。

```
reference: PC = 0xbfc0038c, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfb00000
mycpu    : PC = 0xbfc00010, wb_rf_wnum = 0x08, wb_rf_wdata = 0x80000000
```

图 7 控制台报错信息

（2） 分析定位过程

从报错位置的前一条指令开始看，程序没有跳转。从 PC 可以判断前一指令为 beq 指令。这一步本来应该发生跳转，但是却没有，于是判断跳转条件没有满足，其中信号赋值错误。

（3） 错误原因

变量 br_bus 和 br_target 位宽均为 32，变量 br_taken 位宽为 1，所以 IF 模块和 ID 模块中对这三个变量的赋值会导致数据丢失。

```
assign {br_taken,br_target} = br_bus;
```

图 8 IF 模块中赋值语句

```
assign br_bus        = {br_taken,br_target};
```

图 9 ID 模块中赋值语句

（4） 修正效果

给 br_bus 增加一位位宽。可以修改 mycpu.h 文件中的宏定义`BR_BUS_WD 为 33.

修改后同一指令处不再报错。

## 7. 错误 7：移位指令 srl 出错

（1） 错误现象

控制台报错，提示 wb_rf_wdata 出错。

```
reference: PC = 0xbfc110e0, wb_rf_wnum = 0x02, wb_rf_wdata = 0xadff20c0
mycpu    : PC = 0xbfc110e0, wb_rf_wnum = 0x02, wb_rf_wdata = 0x2dff20c0
```

图 10 控制台报错信息

（2） 分析定位过程

先看 PC 值，确认此时是移位指令，也就是说此时 wb_rf_wdata 为 alu_result，ALU 模块中移位操作出错。

（3） 错误原因

ALU 模块中，移位指令的计算结果只保留了低 31 位，导致出错。

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

图 11 移位指令赋值出错

（4） 修正效果

将上图中 30 改为 31。修改后同一指令处不再报错。

## 8. 错误 8：越沿赋值

（1） 错误现象

这一错误在此任务中不会影响结果，但是为了代码规范，防止以后出错，所以必须修改。

（2） 分析定位过程

在检查 MEM 模块时，发现给信号 es_to_ms_bus_r 赋值时采用的是阻塞赋值（图中光标处）。

```
always @(posedge clk) begin
    if (reset) begin
        ms_valid <= 1'b0;
    end
    else if (ms_allowin) begin
        ms_valid <= es_to_ms_valid;
    end

    if (es_to_ms_valid && ms_allowin) begin
        es_to_ms_bus_r  = es_to_ms_bus;
    end
end
```

图 12 阻塞赋值

（3） 错误原因

没有采用非阻塞赋值。

（4） 修正效果

修改为非阻塞赋值。

## 四、实验总结（可选）

修改所有错误后，上板成功（因为绿色 led 灯亮度太高，所以，为了看清显示的数字，调低了整张照片的亮度）。

错误太多了，我裂开了.jpg。

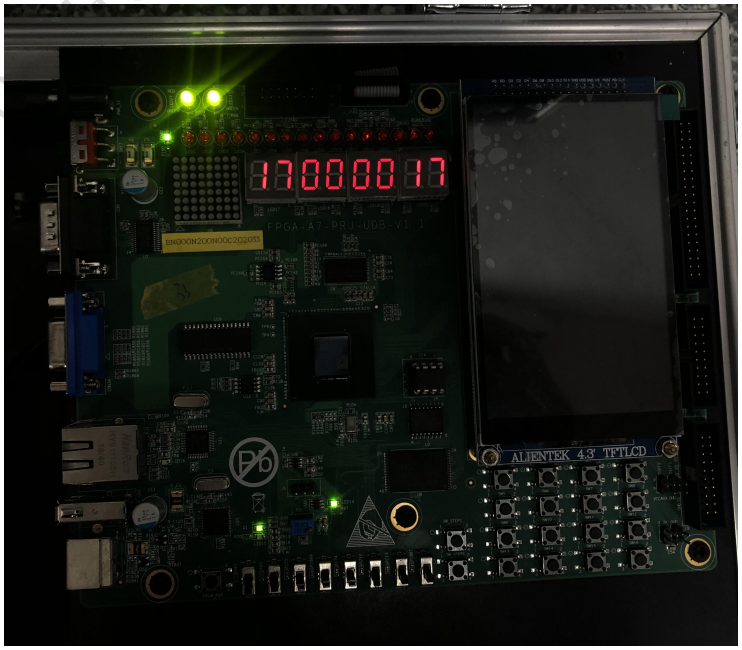
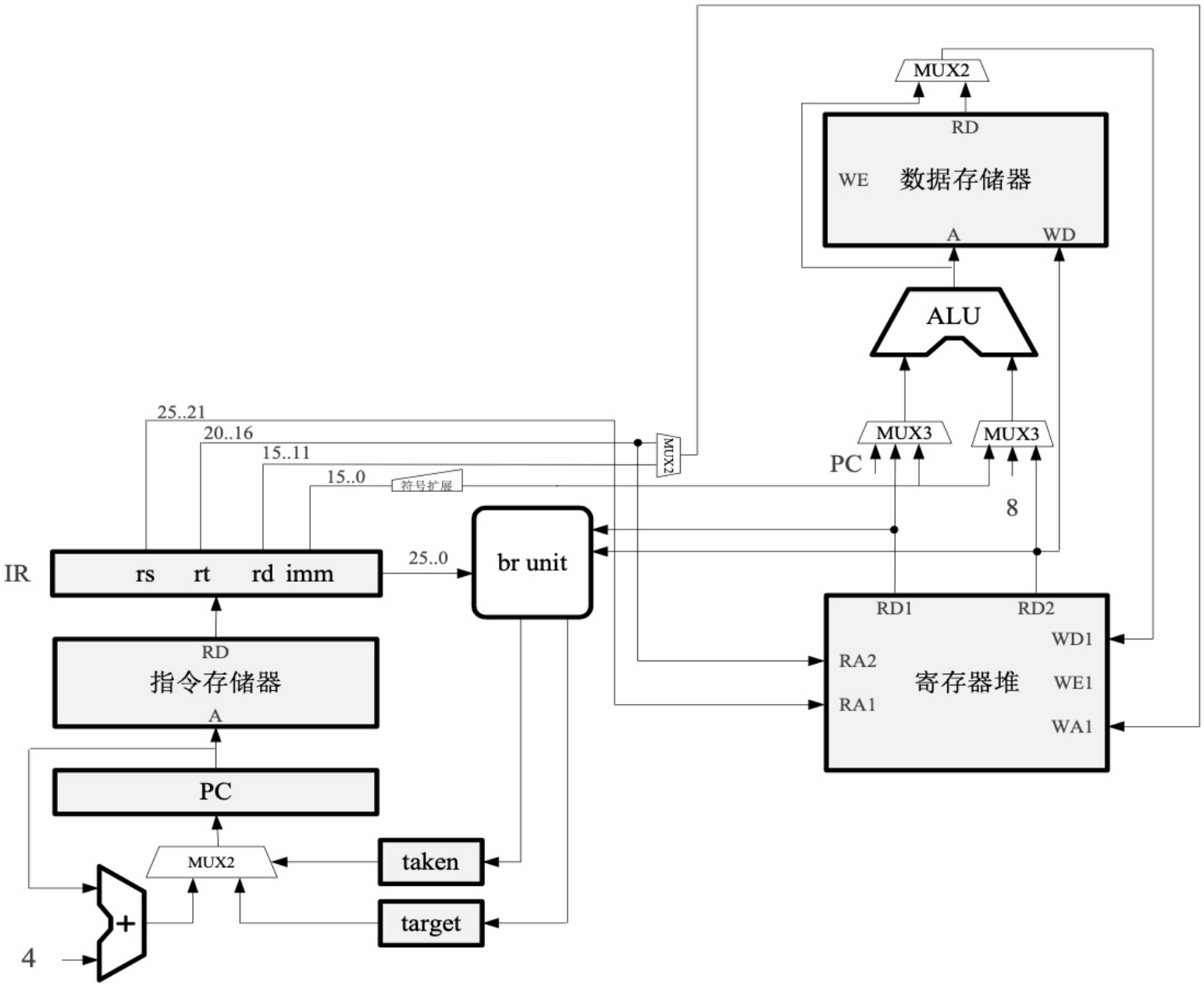



图 13 上板成功结果


MUX2
RD
WE
数据存储器
A
WD
ALU
MUX3
MUX3
PC
8
25..21
20..16
15..11
15..0
符号扩展
MUX2
IR
rs
rt
rd
imm
25..0
br unit
RD1
RD2
WD1
RA2
寄存器堆
WE1
RA1
WA1
RD
指令存储器
A
PC
MUX2
taken
target
+
4