实验 9 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 lab8 的前提下添加了对中断的处理和其他例外处理，主要在不同流水级添加了 excode 的处理，新增加 break 指令，并在最后的写回级报出中断例外信号给 ID 级。

二、实验设计（40%）

（一）总体设计思路

本次的 CPU 在 lab8 的基础上添加了其他例外处理，包括地址错，整数溢出，保留指令例外支持，时钟中断等。由于大致框架已经在 lab8 中实现，所以具体到 lab9 实验中涉及的改动并不多，主要是增添了一部分小细节的实现。 具体涉及的分模块设计思路如下：

1. IF 模块设计思路：增添本条指令的虚拟地址（实际就是当前 pc）
2. ID 模块设计思路：例行对新指令译码（lab9 中只有 break 指令），增加了 ID 阶段的错误代码，包括 syscall, break 和 reserved_inst 三种例外以及 WB 模块传来的中断信号。
3. EXE 模块设计思路：该阶段例化的 alu 添加了 overflow 信号的判断，同时增加了 EXE 阶段的错误代码判断，包括了 ov，adel,ades 等三种例外，并对 badvaddr 进行了逻辑选择
4. MEM 模块设计思路： 在总线中额外添加了 badvaddr 的 32 位数据，向下一级流水，其他地方无改动。（由于 mem 级不会再有新的例外出现）
5. WB 模块设计思路：根据此阶段读出来的 cp0 寄存器数据，做出中断处理。同时此阶段例化的 cp0 寄存器中，额外添加了 cp0_badvaddr 的逻辑处理部分。
6. Alu 模块设计思路：使用了位运算的方式（其实更接近选择器）增添了 overflow 信号

（Mark:在之后的分模块设计思路中 alu 的设计变动并入 exe 模块设计中列出。）

最后的总体设计思路如下图 1 所示。

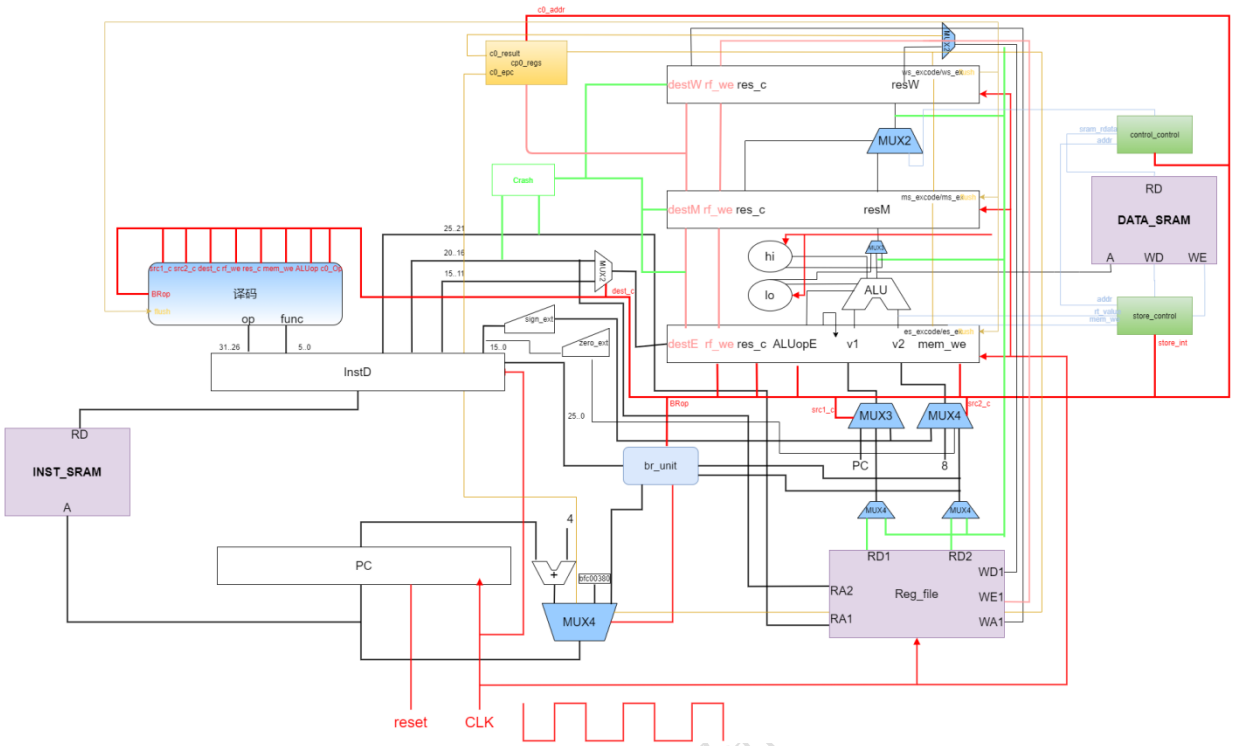


图 01: 整体数据通路设计

（二）重要模块 1 设计：IF 模块

1、工作原理

根据上一条指令最后的 WB 传出信号，决定新 PC 取值，同时处理取值阶段可能出现的例外，以及延迟槽信号判断

2、接口定义（仅展示 lab8 更新）

名称	方向	位宽	功能描述
fs_to_ds_bus	OUT	103	到译码级流水，新增 badvaddr 地址

3、功能描述

由于在 Lab8 中已经实现了 IF 阶段对取值 PC 错误例外的判断，因此 IF 阶段相对 lab8 没有进一步的例外信息处理，主要差别在于向下一级流水的总线中添加了新的 32 位信号，如下图 01 所示：

```
assign badvaddr_value = fs_pc;
assign fs_to_ds_bus = {badvaddr_value,//102:71
                       fs_excode, //70:66
                       fs_ex   ,   //65:65
```

图 01：IF 阶段的 badvaddr 地址信息

（三）重要模块 2 设计：ID 模块

1、工作原理

（lab9 新增）增加了 ID 阶段的错误代码，包括 syscall, break 和 reserved_inst 三种例外，并加入中断信号。

2、接口定义（仅展示 lab9 更新）

名称	方向	位宽	功能描述
ds_to_es_bus	OUT	211	向 exe 的流水，具体增添部分见 3 的功能描述
Interrupt	IN	1	中断信号，来自 WB 阶段指令
fs_to_ds_bus	IN	103	到译码级流水，新增 badvaddr 地址

3、功能描述

相对于 lab8, lab9 需要考虑 ID 阶段更多的出错情况，所以更新后的例外信号处理如下图 02 所示：

```
assign ds_ex = ds_valid ? ( inst_syscall | fs_ex | inst_break | reserved_inst | interrupt): 1'b0;
assign ds_excode = ds_valid ? (interrupt ? 5'h0  :
                               fs_ex ? fs_excode :
                               ds_excode_temp) : 5'b0;
```

图 02：译码阶段例外信号处理

上图中的 ds_excode_temp 是根据错误信息预处理的错误代码，具体实现方式如下图 03 所示：

```
assign inst_valid   = inst_add | inst_addi | inst_addiu | inst_addu | inst_and
                      inst_mult | inst_multu | inst_nor | inst_or | inst_ori | in
                      inst_slt | inst_slti | inst_sltiu | inst_sltu | inst_sub |
                      inst_bne | inst_beq | inst_bgez | inst_bgezal | inst_bgtz |
                      inst_break | inst_eret | inst_syscall | inst_mfc0 | inst_mt
                      inst_lw | inst_lb | inst_lbu | inst_lh | inst_lhu | inst_lw
assign reserved_inst = ~inst_valid;
assign ds_excode_temp = inst_syscall ? 5'h08 :
                        inst_break   ? 5'h09 :
                        reserved_inst? 5'h0a : 5'b0;
```

图 03：译码阶段 exccode 预处理过程

上图中 inst_valid 集合了目前所有已有的指令集合（由于过长就没有截图完全），当进入的指令并没有进入上述有效指令的集合之后就认为发生了保留指令例外错误。

（四）重要模块 3 设计：EXE 模块（仅描述 lab9 更新）

1、工作原理

Lab9 中，exe 模块需要判断地址错误和计算溢出错误，同时要对 badvaddr 内容更新（根据地址错是哪种地址错）

2、接口定义

名称	方向	位宽	功能描述
ds_to_es_bus	IN	211	来自 ID 的流水，传递了 ov 指示信号等

3、功能描述

对 EXE 阶段可能产生的错误的判断如下图 04 所示:

```
assign  exception_overflow = inst_overflow & overflow;
assign  es_adel = (inst_lw & (addr_low != 2'b00)) |
                  (inst_lh & addr_low[0]) |
                  (inst_lhu & addr_low[0]);
assign  es_ades = (inst_sw & (addr_low != 2'b00)) |
                  (inst_sh & addr_low[0]);
assign  es_excode_temp = exception_overflow   ? 5'h0c :
                         es_adel              ? 5'h04 :
                         es_ades              ? 5'h05 : 5'b0 ;
```

图 04：EXE 阶段例外代码的预处理

Adel 错误发生在在指令取值/load 类指令地址非对齐，ade 错误发生在 store 指令地址非对齐的时候，而 ov 错误发生在可能发生 ov 的三种指令（单纯的加法和减法）以及 alu 报错的情况

在 lab9 阶段更复杂的例外判断下的 EXE 阶段例外信号处理：

```
assign  es_we =  es_valid ? (~flush & ~ms_ex & ~ms_eret & ~es_ex):1'b0 ;
assign  es_ex =  es_valid ? (ds_ex | exception_overflow | es_ades | es_adel) : 1'b0;
assign  es_excode = es_valid ? (ds_ex ? ds_excode : es_excode_temp):5'b0 ;
```

图 05：EXE 阶段例外信号处理

在 EXE 阶段对 bad_vaddr 的值进行选择如下图 06 所示：

```
assign  badvaddr_value = (ds_ex && ds_excode == 5'h04) ? PC_badvaddr : es_alu_result;
```

图 06：badvaddr 的选择器

按照讲义要求，当错误为读地址错误时，取得值就是原值，但是如果导致地址出错的原因是地址非对齐，那 BadVAddr 的低两位要严格的更新为非对齐地址的低两位。（表现出来就是用 alu_result 的值代替）

（由于 ALU 部分单独改动较少，故 alu 改动并入 EXE 阶段展示）

ALU 部分添加 ov 处理如下图 07 所示：

```
assign overflow   = (op_add & ~(alu_src1[31]) & ~(alu_src2[31]) &  add_sub_result[31])
                  | (op_add &  (alu_src1[31]) &  (alu_src2[31]) & ~add_sub_result[31])
                  | (op_sub & ~(alu_src1[31]) &  (alu_src2[31]) &  add_sub_result[31])
                  | (op_sub &  (alu_src1[31]) & ~(alu_src2[31]) & ~add_sub_result[31]);
```

图 07：overflow 的位运算结果

（五）重要模块 4 设计：WB 模块

1、工作原理

本次实验中 WB 阶段需要给出中断信号，在例化 cp0.v 模块的同时完成中断判断

2、接口定义（仅展示 lab9 更新）

名称	方向	位宽	功能描述
es_to_ms_bus	IN	130	EXE 阶段流水过来的数据，主要增加了 bad_vaddr
ms_to_ws_bus	OUT	123	Mem 阶段向下一级的流水，主要增加了 bad_vaddr

名称	方向	位宽	功能描述
interrupt	OUT	1	向外输出（报错）中断信号

3、功能描述

C0_result 的选择器增加了 badvaddr 部分：

```
assign c0_result = (c0_addr == `CR_STATUS)   ? c0_status   :
                   (c0_addr == `CR_CAUSE)    ? c0_cause    :
                   (c0_addr == `CR_EPC)      ? c0_epc      :
                   (c0_addr == `CR_COMPARE)  ? c0_compare  :
                   (c0_addr == `CR_COUNT)    ? c0_count    :
                   (c0_addr == `CR_BADVADDR) ? c0_badvaddr :32'b0;
```

图 08：c0_result 的位运算结果

例化 cp0 模块的时候增加了两个输入输出信号：

wb_badvaddr(badvaddr_value)，和.count_eq_compare(count_eq_compare)

对于中断（这里主要是时钟中断）的判断如下图所示：

```
assign c0_status_im  = c0_status [15:8];
assign c0_cause_ip   = c0_cause  [15:8];
assign c0_status_exl = c0_status [1]   ;
assign c0_status_ie  = c0_status [0]   ;
assign interrupt = (count_eq_compare ||((c0_cause_ip & c0_status_im) != 8'b0) )
                    & c0_status_ie & ~c0_status_exl & ws_valid ;
```

图 09：中断信号的处理

需要同时满足开中断条件和时钟中断或外界其他中断的条件才会使得中断信号置 1。

（六）重要模块 5 设计：cp0 模块

1、工作原理

在 lab8 基础上添加了时间中断判断条件和 badvaddr 的传入传出

2、接口定义

名称	方向	位宽	功能描述
Wb_badvaddr	IN	32	Badvaddr 地址信息
C0_badvaddr	OUT	32	对应的 count 寄存器信息
Count_eq_compare	OUT	1	检测时钟中断是否有效

3、功能描述

c0_badvaddr 信号的处理如下：

```
always@(posedge clk)
begin
    if(wb_ex && (wb_excode == 5'h04 || wb_excode == 5'h05))
    begin
        c0_badvaddr_r <= wb_badvaddr;
    end
end
assign c0_badvaddr = c0_badvaddr_r;
```

图 10: c0_badvaddr 地址信息

时钟中断的触发处理如下：

```
assign count_eq_compare = (c0_compare == c0_count);
```

三、实验过程（50%）

（一）实验流水账

11.3 下午开始写，由于框架搭建比较完善 大概 3 小时完成任务

11.5-11.6 陆续利用零碎时间完成实验报告，总用时大概 3 小时

11.5 晚上， 上板通过记忆游戏测试

（二）错误记录

1、错误 1：bad_vaddr 处理不当

（1） 错误现象

PC 写地址正常但是数据不对。

```
[1688377 ns] Error!!!
    reference: PC = 0xbfc68e60, wb_rf_wnum = 0x16, wb_rf_wdata = 0x800cfde1
    mycpu    : PC = 0xbfc68e60, wb_rf_wnum = 0x16, wb_rf_wdata = 0xbfc68e4c
```

图 11: badvaddr 错误信息

（2）分析定位过程

查看 test.s 发现是 mfc0 指令，取得是 badvaddr 寄存器的值，这里发现写回的值应该是写地址出错修改的 badvaddr，不是一个 PC 值，反查代码，发现在 exe 阶段忘记对 badvaddr 进行选择，导致写入的一直只有出错的 PC 值而没有及时更新为需要写的出错的地址。

（3）错误原因

exe 阶段忘记对 badvaddr 进行选择，

（4）修正效果

更改为如图 6 所示的选择器，然后正常跑过该测试点（配图略）

2、错误 2：仿真在某个阶段循环进行

（1） 错误现象

PC 在同一个值循环。

```
[2152000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2162000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2172000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2182000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2192000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2202000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2212000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2222000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2232000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2242000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2252000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2262000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2272000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
[2282000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
```

图 12: 仿真在某个点循环的现象

（2）分析定位过程

查看波形发现反复在下面两条指令之间跳转（见图 13），而这两条指令之前修改了 count 寄存器的值

```
108317   bfc5fa88:   40824800    mtc0    v0,$9
108318   bfc5fa8c:   1000ffff    b   bfc5fa8c <n81_ti_ex_test+0x5c>
108319   bfc5fa90:   00000000    nop
```

图 13: test.S 中的汇编代码

在这里应该会进入时钟中断，否则就会跳入一个死循环一直等待时钟中断触发，所以转而去检查中断信号，最后发现：中断信号正常拉高，但在 ID 阶段的 ex 信号却没有正常拉高，发现没有把中断信号添加进 ex 信号的判断逻辑中。

（3）错误原因

没有把中断信号加入 id 阶段的 ex 信号的判断逻辑。

（4）修正效果

修改后逻辑如下：

```
assign ds_ex = ds_valid ? ( inst_syscall | fs_ex | inst_break | reserved_inst | interrupt): 1'b0;
```

图 14: ds_ex 的逻辑中需要额外添加的信号

之后正常通过该验证点。

3、错误 3：软件中断没有正常触发

（1） 错误现象

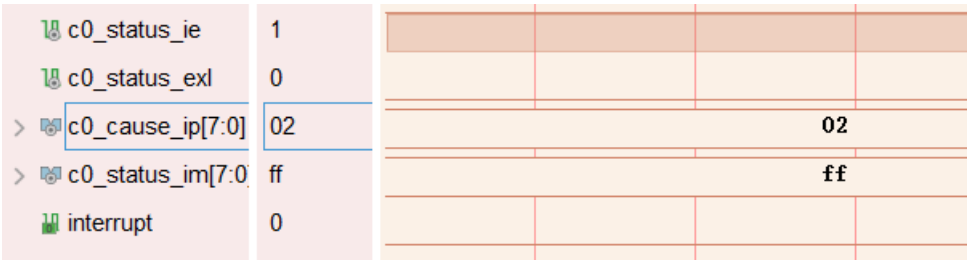
PC 在同一个值循环。

```
-[1755625 ns] Number 8'd81 Functional Test Point PASS!!!
    [1762000 ns] Test is running, debug_wb_pc = 0xbfc16c58
    [1772000 ns] Test is running, debug_wb_pc = 0xbfc16c58
    [1782000 ns] Test is running, debug_wb_pc = 0xbfc16c58
    [1792000 ns] Test is running, debug_wb_pc = 0xbfc16c58
    [1802000 ns] Test is running, debug_wb_pc = 0xbfc16c58
```

图 15: 仿真在某个点循环的现象

（2）分析定位过程

查看 test.s 发现此处测试的是软件中断，没有正常触发，拖出 interrupt 信号发现输入信号值是正确的，但没有



得到想要的结果

图 16: 查看波形的时候发现出现了信号命名错误的截图

（3）错误原因

此处犯得错误较多，包括信号名字错误，没有注意到不等于的优先级比&高等等

（4）修正效果

修改后逻辑如下：

```
assign interrupt = (count_eq_compare ||((c0_cause_ip & c0_status_im) != 8'b0) )
                    & c0_status_ie & ~c0_status_exl & ws_valid ;
```

图 17: 用括号强制匹配后的中断逻辑

之后正常通过该验证点。

四、实验总结（可选）

·

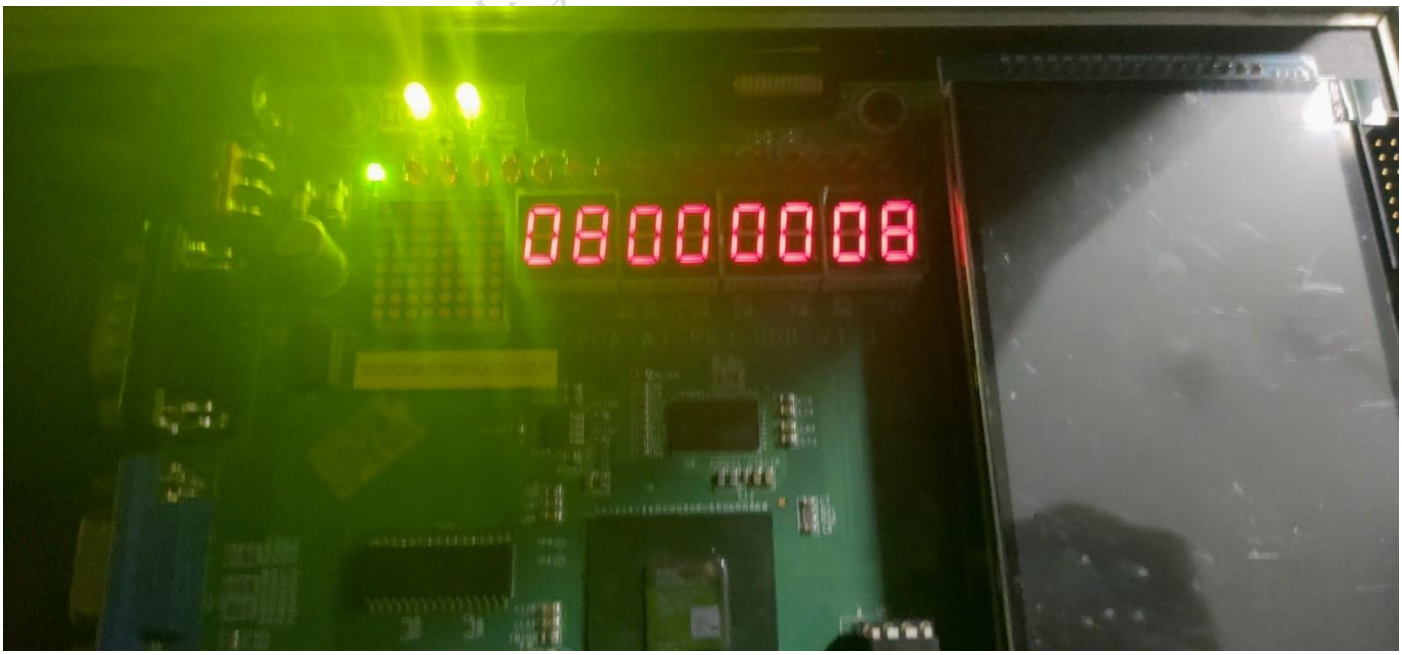


图 18 上板通过记忆游戏测试并获得满分截图

本次实验主要在上一个实验的框架上实现，得益于上一次框架搭建的比较完善，代码写起来比较顺畅，但在一些细节的实现上，比如在实验错误中提到的两个地方的错误，就做的不是很好，一开始设计的时候没有从一个整体的角度思考，导致后期 debug 花费了一定的时间。

以及不等于的优先级这么低是没想到的，这个问题还是提示了基础知识的重要性，这点没有注意到导致 debug 上浪费了很多时间。